



兰州大学

本科毕业论文

论文题目（中文） 基于访谈文本数据建模的抑郁症识别

关键影响因素研究

论文题目（英文） Depression Detection Key Influencing Factors

Study Based on Interview Textual

Data Modeling

学生姓名 夏生杰

指导教师 刘振宇

学 院 信息科学与工程学院

专 业 计算机科学与技术

年 级 2021 级

兰州大学教务处

基于访谈文本数据建模的抑郁症识别关键影响因素研究

中文摘要

随着全球心理健康问题日益受到关注，抑郁症已成为影响人类生活质量的首要精神疾病之一。而传统临床诊断依赖于主观面谈与量表评估，存在耗时、成本高、易受主观偏差影响等局限性，导致大量潜在病例被延误诊断。近年来，随着自然语言处理与机器学习技术的突破，基于文本数据的抑郁症识别逐渐成为研究热点。人类语言作为心理状态的镜像载体，其词汇选择、语法结构、情感倾向等特征往往潜藏着抑郁情绪的蛛丝马迹。通过分析社交媒体文本、临床访谈记录或日常写作等文本数据，研究者试图构建客观、可量化的抑郁检测模型。然而，当前文献过度聚焦于分类模型的性能提升，普遍忽视了对抑郁识别影响因素的关注。此外，经临床验证的开源中文数据集严重匮乏也阻碍了中文领域的抑郁症识别研究。为了探究中文抑郁识别任务中语义单元与数据标签的内在映射情况，本文开展了基于访谈文本数据建模的抑郁症识别关键影响因素研究，主要内容如下：

(1) 为了避免重复性医疗数据收集的隐私问题与资源消耗，本文使用了兰州大学普适感知与智能系统实验室从合作医院采集的标准化临床访谈录音，通过专业的语音识别转写与人工标注构建了中文访谈文本数据集 DepInter-CN。经过统计分析发现，数据集是平衡的，文本长度主要集中在 150 字内，这为将来进行实验提供了参考。

(2) 为了在众多的文本特征构造方法中确定有效方式、构建高性能的模型及进一步探究访谈问题与抑郁识别的关联程度，本文分别从特征工程、模型架构、数据选用三个层面控制变量进行影响讨论。在此之前本文通过选用经典特征构造集与分类模型，筛选出可用于后续实验的基线变量。考虑到非本文主要探究的其他因素也可能与以上三个因素间存在相互影响，因此通过排除归纳法排除性能明显较差的自变量，最终提出了一种融合预训练语言模型、机器学习与深度学习的混合识别架构 DepHybrid，相较其他架构显著提高了识别性能，取得了 77% 的准确率和 F1 分数。

(3) 为了探究架构中的多参数寻优以提升其性能，本文通过将其抽象为一个多臂老虎机问题，引入了 UBC 算法实现自动超参数优化，较原先准确率提高了 0.23%，召回率提升了近 10.3%，AUC 提升了近 4.5%。

(4) 为了提高人工智能技术在医疗场景中的透明度，本文使用了 t-SNE 与 SHAP 方法，分别从宏观特征空间与微观特征贡献度验证了模型架构的可信度。

关键词：抑郁识别；访谈文本数据；深度学习；控制变量法；混合架构

Depression Detection Key Influencing Factors Study Based on Interview Textual Data Modeling

Abstract

As the world pays more attention to mental health issues, depression has become one of the leading psychiatric disorders affecting quality of life. Traditional clinical diagnoses rely on subjective interviews and scale assessments, which are time-consuming and costly. They are also susceptible to subjective bias, resulting in delayed diagnoses for a large number of potential cases. In recent years, natural language processing and machine learning technology have made significant breakthroughs, and depression recognition based on text data has gradually become a research hotspot. Human language, as a carrier of psychological state, often contains traces of depressive mood in features such as vocabulary selection, grammatical structure, and emotional tendency. By analyzing textual data, such as social media posts, clinical interview transcripts, and personal writings, researchers have attempted to develop objective, quantifiable models for detecting depression. However, current literature excessively focuses on improving the performance of classification models and generally ignores factors affecting depression recognition. Additionally, the severe lack of clinically validated, open-source Chinese datasets hinders depression recognition research in the Chinese domain. To explore the intrinsic mapping of semantic units and data labels in the Chinese depression recognition task, this thesis studied the key factors influencing depression recognition based on interview text data modeling.

(1) To avoid privacy issues and the consumption of resources associated with repetitive medical data collection, this thesis used standardized clinical interview recordings collected by the Ubiquitous Awareness and Intelligent Solutions Laboratory at Lanzhou University from partner hospitals. The thesis then constructed a Chinese interview text dataset, DepInter-CN, through professional speech recognition transcription and manual annotation. After statistical analysis, the thesis found that the dataset is balanced and that the text length is mainly concentrated within 150 words. This provided a reference for future experiments.

(2) To determine an effective method among many text feature construction methods, build a high-performance model, and further explore the correlation between interview questions and depression recognition, this thesis discussed the influence of control variables at three levels: feature engineering, model architecture, and data selection. Prior to this study, baseline variables for subsequent experiments were screened by selecting classical feature construction sets and classification

models. Considering that other factors not discussed in this thesis may interact with the three aforementioned factors, the independent variables with significantly poorer performance were excluded using the exclusion induction method. Finally, a hybrid recognition architecture, DepHybrid, was proposed. DepHybrid integrates a pre-trained language model, machine learning, and deep learning, achieving an accuracy rate of 77% and an F1 score, outperforming other architectures.

(3) To explore multi-parameter optimization in architecture and enhance performance, this thesis introduced the UBC algorithm. It achieved automatic hyper-parameter optimization by abstracting the problem as a multi-armed slot machine, improving accuracy by 0.23%, recall by 10.3%, and AUC by 4.5% compared to the original.

(4) To improve transparency in the use of AI technology in healthcare, this thesis used t-SNE and SHAP methods to validate the credibility of the model architecture in terms of macroscopic and microscopic feature spaces, respectively.

Keywords: Depression detection; Interview text data; Deep learning; Control variable approach; Hybrid architecture

目 录

中文摘要	I
英文摘要	II
第一章 绪 论	1
1.1 研究背景及意义	1
1.2 国内外研究现状	2
1.2.1 抑郁症状定义	2
1.2.2 抑郁识别研究	2
1.2.3 基于文本数据的抑郁识别研究	3
1.2.4 面临挑战	6
1.3 主要工作	6
1.3.1 研究目的	6
1.3.2 研究内容	7
1.3.3 主要贡献	7
1.4 文章结构安排	9
1.5 本章小结	10
第二章 相关理论及技术介绍	11
2.1 引言	11
2.2 机器学习相关理论	11
2.2.1 统计学习方法	11
2.2.2 在线学习方法	14
2.2.3 集成学习方法	14
2.2.4 自动机器学习	17
2.3 深度学习相关理论	17
2.3.1 前馈神经网络	18
2.3.2 反馈神经网络	19
2.3.3 序列到序列模型	19
2.3.4 图神经网络	21
2.4 文本表征技术概述	21
2.4.1 词袋模型	21
2.4.2 词嵌入	22
2.4.3 中文分词	22
2.5 迁移学习相关理论	22

2.6 强化学习相关理论	23
2.7 模型性能评价体系	23
2.8 本章小结.....	24
第三章 访谈文本数据集构建与分析	25
3.1 引言	25
3.2 数据集构建.....	25
3.3 数据内容统计分析	26
3.4 本章小结.....	28
第四章 基于文本数据的抑郁识别关键影响因素分析	29
4.1 引言	29
4.2 模型提出.....	29
4.2.1 DepHybrid 架构.....	29
4.2.2 UCB 参数优化算法	32
4.3 实验目的.....	33
4.4 实验设计.....	33
4.4.1 文本特征类型设计	35
4.4.2 神经网络结构设计	37
4.4.3 实验环境	49
4.4.4 实验流程	50
4.5 实验分析.....	52
4.5.1 数据预处理	52
4.5.2 预实验	52
4.5.3 优秀特征构建模型的筛选实验	59
4.5.4 基于 BoW、BERT、RoBERTa 和 ERNIE 特征的模型设计选择实验	61
4.5.5 访谈问答数据选用对比实验	71
4.5.6 案例分析实验	73
4.6 实验结论.....	77
4.7 本章小结.....	78
第五章 总结与展望	79
5.1 总结	79
5.2 未来展望.....	79
参考文献	81
附 录	88
A. 1 抑郁症文本识别系统设计	88
A. 1. 1 需求分析	88

A. 1. 2 系统设计方案	89
A. 1. 3 系统展示	92
A. 1. 4 系统设计小结	94
A. 2 实验环境配置	95
A. 2. 1 Windows 系统	95
A. 2. 2 Ubuntu 子系统	95
A. 3 实验核心源代码	95
A. 3. 1 DepHybrid 抑郁识别架构	95
A. 3. 2 基于 ERNIE-LR 的 UCB 参数优化方法与其他优化方法的对比实验	100
A. 3. 3 Streamlit 抑郁文本识别系统	104
A. 4 基于 ERNIE 语义的实验复现源代码	106
A. 4. 1 TextCNN	106
A. 4. 2 CapsNet	109
A. 4. 3 BertGCN	113
A. 4. 4 HBGA	119
A. 4. 5 A-Hybrid	125
A. 4. 6 M-BERT	128
A. 4. 7 CBA	133
A. 4. 8 AMCDD	136
A. 5 参考书目	138
致 谢	139

图 目 录

图 1.1 二维情感空间分布	2
图 1.2 文本数据的抑郁识别研究的发展历程	6
图 1.3 本文的结构框架	9
图 2.1 Stacking 方法	15
图 2.2 Bagging 方法	16
图 2.3 Boosting 方法	17
图 2.4 前馈神经网络	18
图 2.5 卷积神经网络	18
图 2.6 反馈神经网络	19
图 2.7 长短期记忆网络	19
图 2.8 变换器模型	20
图 2.9 图神经网络	21
图 3.1 访谈数据集文本长度及数据项分布	27
图 3.2 访谈数据集词云图	28
图 4.1 DepHybrid 架构组件图	31
图 4.2 SFC 模块, 其中 \mathcal{S} 为特征维数	37
图 4.3 FC-Drop 模块, 其中 \mathcal{S} 为特征维数, \odot 表示元素乘, θ 为丢弃隐藏单元概率, σ 为 sigmoid 函数	38
图 4.4 Conv-Pool 模块, 其中 C_{in} 为特征通道数, n 为滤波器个数, L 表示时间序列长度	39
图 4.5 CAM 模块, 其中 F 为特征通道数, \odot 表示元素乘, L 表示序列长度	40
图 4.6 Flatten 模块示意图	41
图 4.7 ResLn 模块, 其中 C_{in} 为特征通道数, F 表示滤波器个数, \mathcal{L} 为损失函数, L 表示序列长度, $K = 3$ 为卷积核大小	42
图 4.8 LSTM 模块, 其中 C_{in} 为特征维度, L 表示时间步长维度	43
图 4.9 BiLSTM 模块, 其中 C_{in} 为特征维度, L 表示时间步长维度	44
图 4.10 GRU 模块, 其中 C_{in} 为特征维度, L 表示时间步长维度	45
图 4.11 BiGRU 模块, 其中 C_{in} 为特征维度, L 表示时间步长维度	46
图 4.12 SAM 模块示意图	47
图 4.13 MSA 模块示意图	48
图 4.14 LayerNorm 模块示意图	49
图 4.15 抑郁识别任务框架	51

图 4.16 抑郁识别任务框架工作流程	51
图 4.17 BERT 嵌入下较优基线模型性能表现.....	55
图 4.18 BoW 下较优基线模型性能表现.....	55
图 4.19 SVM 模型表现与 BERT 嵌入最大标记长度的关系	56
图 4.20 LR 模型表现与 BERT 嵌入最大标记长度的关系	57
图 4.21 SFC 模块表现与 BERT 嵌入最大标记长度的关系	57
图 4.22 Extra tree 模型表现与 BERT 嵌入最大标记长度的关系	58
图 4.23 RF 模型表现与 BERT 嵌入最大标记长度的关系.....	58
图 4.24 MLP 模型表现与 BERT 嵌入最大标记长度的关系	59
图 4.25 各访谈问题的综合性能趋势与变异范围	72
图 4.26 各访谈问题对于抑郁识别性能的贡献情况分析	72
图 4.27 数据集训练前后的 ERNIE 特征可视化分布图.....	74
图 4.28 第一个样本的文本解释图	76
图 4.29 第二个样本的文本解释图	76
图 4.30 第三个样本的文本解释图	76
图 4.31 第四个样本的文本解释图	76
图 4.32 第五个样本的文本解释图	77
图 A.1 抑郁症文本识别系统设计方案	89
图 A.2 抑郁症文本识别系统组件图	90
图 A.3 抑郁症文本识别系统类图	90
图 A.4 抑郁症文本识别系统活动图	91
图 A.5 抑郁症文本识别系统用例图	91
图 A.6 抑郁症文本识别系统界面	92
图 A.7 系统分析界面	93
图 A.8 系统底部分析界面	93
图 A.9 系统详细分析数据界面	94
图 A.10 系统侧边栏隐藏后界面	94

表 目 录

表 1.1 常用的抑郁评分量表.....	3
表 2.1 混淆矩阵结构示例	23
表 3.1 访谈文本数据集关键主题内容	25
表 3.2 受访者样本信息分布情况	26
表 3.3 访谈文本数据集示例.....	26
表 4.1 词袋模型提取特征详细描述	35
表 4.2 预训练语言模型类型详细描述	36
表 4.3 其余参数设置	50
表 4.4 基于 BoW 的访谈文本抑郁症识别基线模型性能总结	53
表 4.5 基于 BERT 嵌入的访谈文本抑郁症识别基线模型性能总结	54
表 4.6 基于 Dense 模块的不同的特征构造策略性能表现.....	60
表 4.7 基于 SFC 模块的不同的特征构造策略性能表现	60
表 4.8 基于 BoW 特征的 FNN 结构模块组合实验结果	61
表 4.9 基于 BERT 嵌入的 FNN 结构模块组合实验结果	62
表 4.10 基于 RoBERTa 嵌入的 FNN 结构模块组合实验结果.....	63
表 4.11 基于 ERNIE 嵌入的 FNN 结构模块组合实验结果	64
表 4.12 基于 BERT 嵌入的 RNN 结构模块组合实验结果	65
表 4.13 基于 RoBERTa 嵌入的 RNN 结构模块组合实验结果	66
表 4.14 基于 ERNIE 嵌入的 RNN 结构模块组合实验结果	67
表 4.15 基于 BoW 特征的模型结构设计性能表现.....	68
表 4.16 基于 BERT 嵌入的模型结构设计性能表现	68
表 4.17 基于 RoBERTa 嵌入的模型结构设计性能表现	68
表 4.18 基于 ERNIE 嵌入的模型结构设计性能表现	68
表 4.19 基于 ERNIE 嵌入的混合结构模块组合实验结果	69
表 4.20 基于 ERNIE 嵌入的 DepHybrid 与现有架构性能表现	70
表 4.21 访谈问题序号与内容映射表	71
表 4.22 访谈问题 4 泛化水平的性能表现	74
表 4.23 对 ERNIE-LR 模型进行参数优化的性能表现	75

专业术语注释表

AdaBoost	Adaptive boosting	自适应增强
Adam	Adaptive moment estimation	自适应矩估计
AI	Artificial intelligence	人工智能
ANN	Artificial neural network	人工神经网络
ASR	Automatic speech recognition	自动语音识别
AUC-ROC	Area Under the Receiver Operating Characteristic Curve	受试者工作特征曲线下面积
AutoML	Automated machine learning	自动机器学习
Bagging	Bootstrap Aggregating	引导聚集(袋装)算法
BART	Bidirectional and auto-regressive transformer	双向自回归转换器
BDI	Beck Depression Inventory	贝克抑郁自评量表
BERT	Bidirectional encoder representations from transformers	基于变换器的双向编码器表示技术
BiLSTM	Bi-directional Long-short term memory	双向长短期记忆网络
BiGRU	Bi-directional Gated recurrent unit	双向门控循环单元
BO	Bayesian Optimization	贝叶斯优化
BoW	Bag-of-words model	词袋模型
BP	Backpropagation	反向传播
CAM	Channel Attention Mechanism	通道注意力机制
CapsNet	Capsule Network	胶囊网络
CART	Classification and regression tree	分类与回归树
CatBoost	Categorical Boosting	类别型梯度提升
CBOW	Continuous Bag-of-words model	连续词袋模型
CCP	Cost-Complexity Pruning	代价复杂度剪枝
CFAPS	Chinese Facial Affective Picture System	中国面孔表情图片系统
CNN	Convolutional neural network	卷积神经网络
Conv	Convolutional	卷积
CQP	Convex quadratic programming	凸二次规划
CPU	Central Processing Unit	中央处理器
CRF	Conditional random field	条件随机场
CV	Computer vision	计算机视觉
CWS	Chinese word segmentation	中文分词
DL	Deep learning	深度学习

DNN	Deep Neural Network	深度神经网络
Drop	Dropout	暂退层
DSM-5	Diagnostic and Statistical Manual of Mental Disorders, Fifth Edition	《精神疾病诊断与统计手册》第五版
DT	Decision tree	决策树
EDD	Experimental Design	实验设计
EEG	Electroencephalogram	脑电图
EL	Ensemble learning	集成学习
ELM	Entity-Level Masking	实体级掩码
ERNIE	Enhanced Representation through Knowledge Integration	通过知识集成实现增强型表示
Extra tree	Extremely randomized tree	极端随机树
FC	Fully Connection	全连接
FCNN	Fully Connected Neural Network	全连接神经网络
FFN	Feedforward neural network	前馈神经网络
FN	False Negative	假阴性
FP	False Positive	假阳性
FPR	False Positive Rate	假阳性率
GABA	γ -aminobutyric acid	γ -氨基丁酸
GAT	Graph attention network	图注意力网络
GBD	Global Burden of Disease Study	全球疾病负担研究
GBDT	Gradient Boosting Decision Tree	梯度提升决策树
GCN	Graph convolutional network	图卷积神经网络
GOSS	Gradient-based one-side sampling	梯度基采样
GPU	Graphics Processing Unit	图形处理器
GLM	Generalized linear model	广义线性模型
GloVe	Global Vectors for Word Representation	带有全局向量的词嵌入
GNN	Graph neural network	图神经网络
GP	Gaussian Process	高斯过程
GPC	Gaussian process classification	高斯过程分类
GPR	Gaussian process regression	高斯过程回归
GPT	Generative pre-trained transformer	基于转换器的生成式预训练模型
GRN	Graph recurrent network	图循环网络
GRU	Gated recurrent unit	门控循环单元
HAM-D	Hamilton Rating Scale for Depression	汉密顿抑郁量表

HGB	Histogram-based Gradient Boosting	基于直方图的梯度提升树
HIS	Hospital Information System	医院信息系统
HMM	Hidden Markov Model	隐马尔可夫模型
HPO	Hyperparameter optimization	超参数优化
ICD-11	International Classification of Diseases 11th Revision	《疾病和有关健康问题的国际统计分类》第十一次修订本
IDS	Inventory of Depressive Symptoms	抑郁症状量表
KKT	Karush–Kuhn–Tucker	卡鲁什-库恩-塔克
k-means	k-means clustering	<i>k</i> 均值算法
k-NN	k-nearest neighbors	<i>k</i> 近邻算法
LayerNorm	Layer Normalization	层归一化
LDA	Latent Dirichlet allocation	隐含狄利克雷分布
LightGBM	Light Gradient Boosting Machine	轻量级梯度提升机
LIME	Local Interpretable Model-agnostic Explanations	局部可解释模型无关解释
LLM	Large language model	大型语言模型
LR	Logistic regression	逻辑斯谛回归
LSTM	Long-short term memory	长短期记忆网络
LTCR	Long-term Cumulative Reward	长期累积奖励
MAB	Multi-armed Bandit	多臂老虎机
MADRS	Montgomery–Åsberg Depression Rating Scale	蒙哥马利-阿斯伯格抑郁症评定量表
MAG	Multimodal Adaptation Gate	多模态适应门
MARL	Multi-agent reinforcement learning	多智能体强化学习
M.I.N.I.	Mini-international neuropsychiatric interview	简明国际神经精神障碍访谈检查
ML	Machine learning	机器学习
MLE	Maximum likelihood estimation	最大似然估计
MLM	Masked language modeling	掩码语言建模
MLP	Multilayer perceptron	多层感知器
MT	Machine translation	机器翻译
MSE	Mean Squared Error	均方误差
MSM	Multi-head Self-Attention Mechanism	多头自注意力机制
NAS	Neural architecture search	神经架构搜索
NBC	Naive Bayes classifier	朴素贝叶斯分类器

NLP	Natural language processing	自然语言处理
NN	Neural network	神经网络
NSP	Next sentence prediction	下一句预测
OHE	One-hot encoding	独热编码
OOB	Out-of-Bag	袋外
OOV	Out-of-vocabulary	未登录词
OS	Operating System	操作系统
PAA	Passive-Aggressive algorithm	被动攻击学习算法
PCA	Principal component analysis	主成分分析
PHQ-9	nine-item Patient Health Questionnaire	九项患者健康问卷
Pool	Pooling	池化
PR	Pattern recognition	模式识别
QIDS	Quick Inventory of Depressive Symptomatology	抑郁症状快速量表
RAM	Random Access Memory	随机存取存储器
RBF	Radial basis function	径向基函数
ReLU	Rectified Linear Unit	线性修正单元
ResLn	Residual learning	残差学习
RF	Random forest	随机森林
RL	Reinforcement learning	强化学习
RNN	Recurrent neural network	循环神经网络
RoBERTa	Robustly optimized BERT pretraining approach	可靠优化的 BERT 预训练方法
SAM	Self-Attention Mechanism	自注意力机制
SD	Standard Deviation	标准差
SDS	Zung Self-Rating Depression Scale	抑郁自评量表
SE	Squeeze-and-Excitation	压缩-激励
Seq2Seq	Sequence to Sequence	序列到序列
SFC	Simple Fully Connected	简单全连接
SGD	Stochastic gradient descent	随机梯度下降
SHAP	Shapley Additive Explanations	沙普利额外解释
SLM	Small language model	小型语言模型
SMO	Sequential minimal optimization	序列最小优化
SNN	Spiking neural network	脉冲神经网络
SNS	Social networking service	社交网络服务
SSD	Solid State Disk	固态硬盘
Stacking	Stacked Generalization	堆叠泛化
SVM	Support vector machine	支持向量机

TAT	Thematic Apperception Test	主题统觉测验
TCN	Temporal Convolutional Network	时域卷积网络
TDM	Text data mining	文本挖掘
TextCNN	Text convolutional neural network	文本卷积神经网络
TF-IDF	Term Frequency-Inverse Document Frequency	词频-逆文档频率
TL	Transfer learning	迁移学习
TN	True Negative	真阴性
TP	True Positive	真阳性
TPR	True Positive Rate	真阳性率
t-SNE	t-distributed stochastic neighbor embedding	t 分布随机领域嵌入
UCB	Upper Confidence Bound	置信区间上界
UI	User Interface	用户界面
WAV	Waveform Audio File Format	波形音频文件格式
WSL	Windows Subsystem for Linux	适用于 Windows 的 Linux 子系统
Word2Vec	Word to Vector	从单词到向量
WWM	Whole-Word Masking	全词遮蔽
XAI	Explainable Artificial Intelligence	可解释人工智能
XGBoost	eXtreme Gradient Boosting	极致梯度提升

第一章 絮 论

1.1 研究背景及意义

抑郁障碍（Depressive disorder），临床常称抑郁症（Depression，词源为拉丁语“Deprimere”，意为“压抑”）^[1]，是以持续性情绪低落及行为抑制为核心特征的精神疾患（Mental disorder）^[2]。该病症具有多维度的破坏性影响，既可损害个体的生理机能与社会功能，也会显著恶化家庭及社区人际关系^[3]。其临床表现除典型的心境障碍和精力衰退外，常伴随睡眠节律紊乱、食欲异常、病理性自罪感等躯体化症状，严重时可诱发自杀行为^[4]。

从人类疾病认知史的角度考察，抑郁症的病理阐释具有深远的文化渊源。古希腊医学奠基者希波克拉底（Hippocrates）基于体液学说（Humorism），将忧郁状态归因于黑胆汁（Melancholia）过量，该术语由希腊词根“melas”（黑色）与“khōlé”（胆汁）复合而成^[5]。与之形成跨文明对话的是，中国古代医学典籍《黄帝内经》系统论述了“郁证”的病机，提出心肝脾三脏功能失调导致情志异常的经典理论^[6]。

在现代社会转型背景下，心理健康议题的显著性持续提升。以“黑狗（Black Dog）”^[7]为代表的抑郁症隐喻在公共话语中广泛传播，加之媒体对公众人物抑郁事件的密集报道，共同推动该疾病成为社会关注焦点。这种关注度的提升与疾病负担的加重形成强烈反差：全球疾病负担研究（GBD）显示，抑郁障碍在2019年已成为非致命健康损失的首要精神疾病源，其疾病总负担位列全球304种疾患第13位^[8]。在中国精神卫生流行病学调查中，成人抑郁障碍终生患病率达6.8%，但治疗率不足10%^[9,10]，揭示出严峻的诊疗缺口。

该疾病的负外部性不仅体现在健康维度，更形成显著的经济耗损。从直接医疗支出来看，抑郁症相关费用在中国居民预期医疗支出中占比14.7%^[11]，是肥胖症医疗成本的近3倍^[12,13]。间接成本方面，欧洲研究显示抑郁症导致的劳动生产力损失占经济总成本的58.7%^[14]，美国同类数据亦达61%^[15]。这种“双重负担”特征对公共卫生体系提出严峻挑战，而我国当前精神科医生仅占医师总数的1.5%^[16]，专业资源的稀缺性进一步加剧诊疗困境。

在临床诊断领域，现行标准主要依赖DSM^[17]和ICD^[18]体系，辅以PHQ系列量表^[19]等评估工具。然而量表间的信效度差异^[20]、诊断者的主观判断偏差以及患者的认知局限^[21]，共同制约着评估的客观性。这种困境催生了生物标记物研究的快速发展，血清素水平异常^[22]、γ-氨基丁酸（GABA）能系统失调^[23]等神经生化指标被逐步纳入诊断视野，但尚未形成金标准^[24]。

人工智能技术的介入为突破传统诊断瓶颈提供了新范式。基于深度学习的多模态融合系统（Multi-modal data fusion）通过整合语音^[25]、视觉^[26]和文本特征^[27]，构建起客观化的抑郁评估模型。特别是社交媒体文本分析技术^[28]，因其非侵入性、可扩展性强等优势，

已成为精神计算领域的前沿方向。目前研究者正探索将大语言模型（LLM）与多智能体强化学习（MARL）相结合^[29]，以期实现更精准的精神状态解析。

1.2 国内外研究现状

1.2.1 抑郁症状定义

抑郁症的临床早期表现呈现多维度病理特征，主要涵盖三个症状群：核心症状群包括持续性心境低落、兴趣缺失及快感缺乏（Anhedonia）；躯体症状群涉及睡眠节律紊乱（失眠/嗜睡）、食欲异常（暴食/厌食）及持续性疲劳；认知症状群表现为注意力缺陷、自我价值感贬损以及焦虑易激惹等^[30]。这些症状若持续 ≥ 2 周且造成显著社会功能损害，即符合临床诊断标准^[31]。

在情感计算理论发展史上，Russell（1980）提出的二维情绪空间模型具有里程碑意义。该模型将情绪状态解构为效价（Valence，积极-消极维度）和唤醒度（Arousal，高激活-低激活维度）两个正交坐标，构建了情感表征的量化框架^[32]。值得注意的是，抑郁症在此模型中被精准定位在第三象限（低效价-低唤醒），这一理论定位为后续研究提供了重要范式。2016年海峡两岸学者的合作研究对此模型进行了创新性扩展，通过构建可视化情感坐标框架（图1.1），实现了情绪障碍的直观空间映射^[33]。

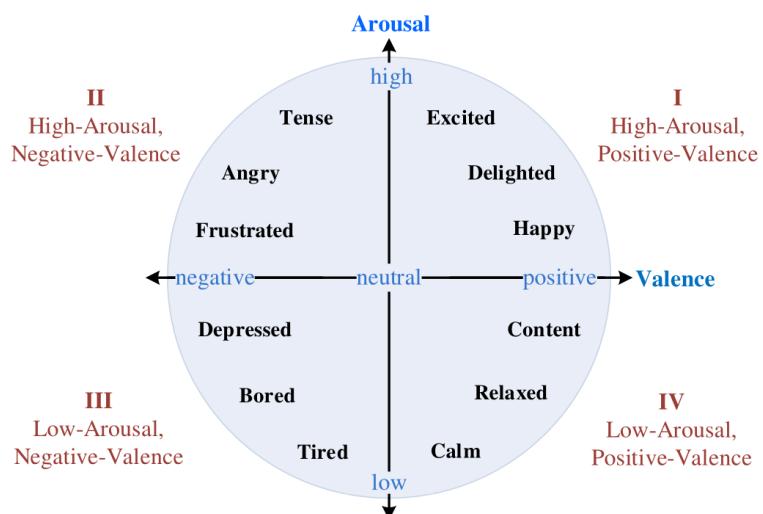


图 1.1 二维情感空间分布^[33]

1.2.2 抑郁识别研究

当前临床实践中，抑郁症的诊断主要遵循双轴评估体系：其一是基于汉密尔顿抑郁量表（HAM-D）等半结构化访谈工具，该量表由 Hamilton 于 1960 年开发，采用改进型李克特量表（Modified Likert Scale）^[34] 进行症状量化评估，将总分划分为五个临床等级：无症状（0-7）、轻度（8-13）、中度（14-18）、重度（19-22）及极重度（ ≥ 23 ）；其二是借助贝克

抑郁量表（BDI）等自评工具，通过 21 项四维度选项设置实现症状强度的梯度测量，总分区间 0-63 对应不同严重程度分层。

表1.1系统梳理了当前主流评估工具，涵盖蒙哥马利抑郁量表（MADRS）、患者健康问卷（PHQ-9）等六大类量表。这些工具与 DSM-5^[30] 及 ICD-11^[18] 诊断标准形成互补，共同构建起多维度的评估框架。然而现有方法存在三重局限：(1) 症状覆盖度不足导致评估盲区；(2) 受试者应答偏差影响信效度；(3) 跨文化适应性缺陷制约诊断普适性。

表 1.1 常用的抑郁评分量表

量表	访谈	自评	项目数量	完成时间（分钟）	发表时间
HAM-D ^[35]	✓		17	20	1960
BDI ^[36]		✓	21	5-10	1961
SDS ^[37]		✓	20	5-10	1965
MADRS ^[38]	✓		10	20-30	1979
IDS ^[39]	✓		30	10-15	1996
PHQ-9 ^[19]		✓	9	<5	2001
QIDS ^[40]	✓		16	5-10	2003

随着计算精神病学（Computational Psychiatry）的兴起，基于多模态生物标记的机器学习方法展现出显著优势。当前研究主要聚焦四大模态：(1) 神经电生理模态：Ay 等^[41] 开发的 CNN-LSTM 模型通过脑电图（EEG）信号分析实现左右脑区 97.66%/99.12% 的分类准确率；(2) 行为表征模态：Alghowinem 团队^[42] 整合言语行为与眼动特征，结合 SVM 分类器提升诊断特异性；(3) 语音文本模态：Sardari 等^[43] 设计的 CNN 自编码器框架使抑郁症语音识别 F1 值提升 7%；(4) 社交媒体模态：李世琪等^[44] 构建的多头注意力模型通过文本特征挖掘达到 92% 分类准确率。

值得注意的是，多模态融合技术正成为领域前沿。Yoon 等^[45] 开发的 D-Vlog 数据集通过交叉注意力机制实现视听特征协同表征，Ansari 等^[46] 则验证了集成模型在跨数据集场景中的优越性能。这些技术突破为构建客观化诊断系统提供了理论支撑。

1. 2. 3 基于文本数据的抑郁识别研究

自然语言处理（NLP）作为连接语言学、认知科学与人工智能的交叉学科，通过构建计算模型实现对语言符号系统的分析、建模与量化解析。在数字医疗革命背景下，NLP 技术正推动精神健康研究从传统症状学向计算精神病学演进，其核心在于建立语言特征与神经精神症状间的可计算映射关系^[47]。这种“数字表型（Digital Phenotyping）”分析范式，通过提取语音韵律、句法复杂度、语义连贯性等语言标记物，为精神障碍的客观诊断提供了新的生物行为指标。

语言数据在精神疾病评估中具有三重不可替代性：(1) 非侵入性：可通过日常交流自

然获取；(2) 高生态效度：反映真实认知状态；(3) 可扩展性：适配社交媒体^[48]、电子病历^[49]等多源数据场景。

1. 探索阶段

现代文本分析技术的起源可追溯至心理学领域的早期探索。精神分析学派创始人弗洛伊德在 1901 年的著作中指出：“口误能泄露个体的潜在意图”^[50]。随后，罗夏等人通过投射测验^[51]，开创了从模糊墨迹反应解读心理特征的范式。麦克利兰团队基于主题统觉测验 (TAT)^[52,53]，系统建立了通过叙事文本分析归属感、权力需求等心理特征的编码体系。这些研究采用专家人工标注方式，对文本中特定词汇和短语进行标记分析。

该阶段研究始于临床观察，研究者注意到抑郁症患者的语言特征变化，尝试通过逻辑推理构建认知模型。受限于当时的技术条件，实验样本普遍较小（通常 10-20 例），主要依赖临床医生的主观判断，具有显著的探索性质。Holtzman 等学者^[54] 虽验证了投射测验的信效度，但对文本能否准确反映抑郁水平仍存疑。

2. 萌芽阶段

20 世纪中期，文本分析开始摆脱特定刺激情境的束缚。Gottschalk 团队^[55,56] 开发的内容分析法，首次实现了对自由谈话文本的弗洛伊德式主题追踪。其方法要求受试者进行 5 分钟自由陈述，通过语法短语分割和专家评分评估心理主题强度。尽管该方法在精神疾病诊断中取得应用^[57]，但因与专家评判标准相关性较低，难以实现计算机化转化。

1960 年代，Stone 等学者^[58,59] 研发的 General Inquirer 系统，首次将麦克利兰的需求编码方案程序化。该系统虽在精神障碍鉴别中展现出价值，但其封闭式算法设计和隐式权重机制限制了推广应用。1980 年代，Weintraub^[60,61] 突破性地发现第一人称单数代词与抑郁水平显著相关，其研究虽被忽视却极具前瞻性。Mergenthaler 于 1996 年开发的 TAS/C 系统^[62]，首次实现心理治疗对话的自动情感分析。

该阶段研究规模显著扩大（样本量 20-30 例），开始采用医学访谈、文字创作等标准化数据采集方式。Oxman 团队^[63] 率先将研究重点转向抑郁识别，通过描述性统计比较患者组间差异。尽管仍以归纳法为主，但计算机辅助程序的引入为后续发展奠定了基础。

3. 起步阶段

1986 年 Pennebaker 的写作疗法研究^[64]，揭示了创伤性写作对心理健康的改善作用。这直接推动其团队于 2001 年开发出划时代的 LIWC (Linguistic Inquiry and Word Count) 文本分析系统^[65]。该系统可自动统计 80 余类心理学相关词汇，包括情感词、自我参照词等关键指标。2004 年 Rude 等学者^[66] 应用 LIWC 发现：抑郁症患者使用负面情感词频率显著高于对照组 ($t=2.37, p=0.02$)，且第一人称单数代词使用量高出 13%。该研究首次证实文本特征在抑郁识别中的诊断价值。

LIWC 的诞生推动了文本分析技术的标准化进程。2000-2010 年间，研究者系统验证了语言模式与心理特征的关系^[67-69]。Chung 等学者^[70] 通过纵向研究证实，LIWC 指标可预测抑郁症状发展，这为后续基于统计学习的研究奠定了基础。

4. 发展阶段

Web2.0 时代社交媒体数据的爆发，推动研究范式向统计学习转变。Moreno 团队^[71]于 2011 年首次证实 Facebook 状态更新中的语言特征与抑郁症状显著相关 ($\exp(B)=2.1$, $p<0.001$)。De Choudhury 等^[28] 构建首个 Twitter 抑郁检测模型，结合 SVM 与多项式回归等方法，实现发病前 6 个月的预测 ($F1=0.63$)。2015 年 Tsugawa 团队^[72] 整合 LDA 主题模型，将日本用户的检测准确率提升至 66%。

该阶段特征工程快速发展：情感词典 (WordNet^[73]、HowNet^[74]) 与词袋模型 (TF-IDF^[75]) 成为主流。Leiva 团队^[76] 通过集成学习融合 SVM、逻辑回归 (LR) 等模型，在 eRisk2017 数据集上取得 16.7% 的性能提升。研究重点正式转向抑郁识别分类问题， $F1$ 值从初期 58% 提升至 63%^[77]。

在这一阶段，情感分析主要依赖于情感词典来进行。情感词典最早用于机器翻译 (MT)，通过提供情感词的情感极性，帮助实现不同粒度下的情感极性划分。现有的大部分情感词典都是人工构建的，如 WordNet^[73]、HowNet^[74]、NTUSD^[78] 等，需要耗费大量的精力和时间。这包括阅读大量的相关资料和现有的词典，以及总结概括含有情感倾向的词语^[79]。此外，还需要对这些词语的情感极性和强度进行不同程度的标注，以确保情感分析的准确性和可靠性^[80]。2013 年，北京邮电大学的 Wang^[81] 等人使用了 HowNet 来计算每条微博的抑郁倾向，基于心理学研究理论的抑郁症患者 10 个特征，开发了一个用于在线心理健康监测的应用。2018 年 Ansari 团队^[46] 使用了 SenticNet^[82] 提取特征以训练混合模型进行抑郁识别。

5. 繁荣阶段

深度学习革命彻底改变了研究范式。2017 年 Transformer 架构的提出^[83]，推动 BERT^[84]、GPT 等预训练模型的应用。Orabi 等^[85] 对比发现，CNN 模型 ($F1=0.86$) 在抑郁检测任务中优于 RNN 模型 ($F1=0.75$)。Zogan 团队^[86] 提出的 DepressionNet 框架，通过融合 CNN 与 BART 模型，在 Twitter 数据集上实现 90.1% 的准确率。

当前研究呈现多模态融合趋势：Haque 等^[87] 提出的 SDCNL 模型，结合无监督标签校正机制，在自杀倾向检测中 AUC 达 0.98。2023 年 Mamba 模型^[88] 通过状态空间建模，推理速度超越 Transformer 5 倍。大语言模型时代，DepRoBERTa^[89] 在 ACL-2022 评测中 $F1$ 值突破 0.64，DeepSeek-R1^[90] 更通过强化学习实现认知推理能力的突破。

该阶段研究维度从单一文本扩展到多模态数据融合。学界共识逐渐形成：需统筹数据质量、特征工程与模型架构的协同优化。当前最优模型不断刷新主要基准测试（如 eRisk、CLPsych）中的指标，标志着技术进入临床实用化前夜。

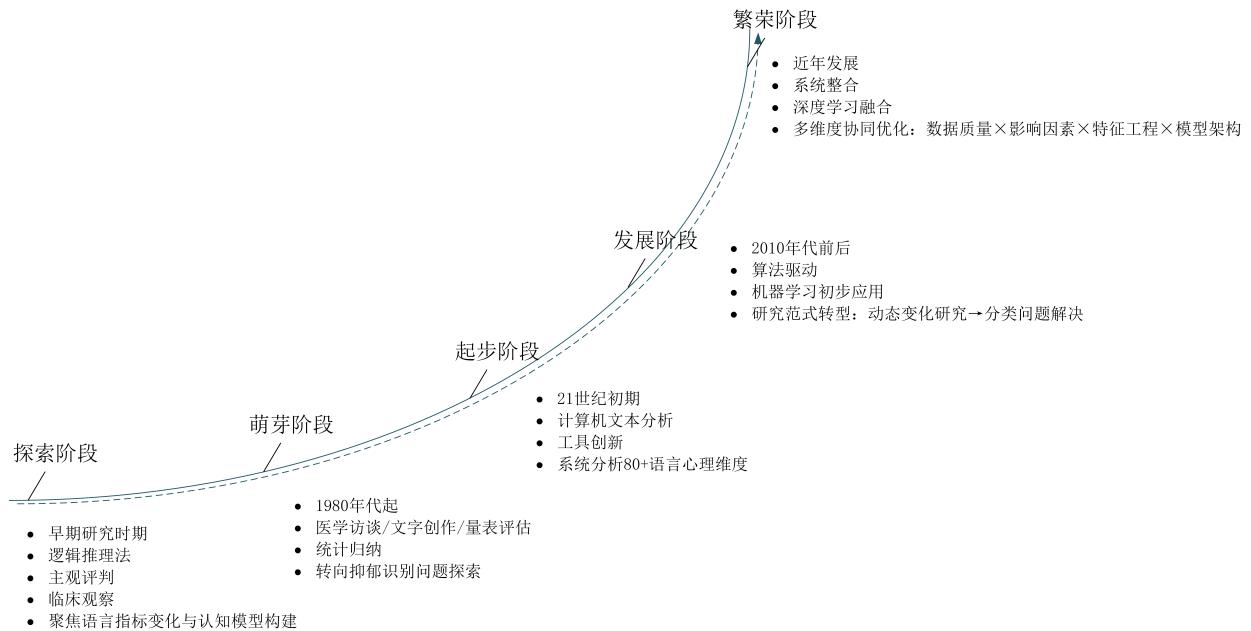


图 1.2 文本数据的抑郁识别研究的发展历程

1.2.4 面临挑战

如图1.2所示，尽管深度学习技术显著提升了文本分析在抑郁症检测中的表现潜力，但现有研究体系仍存在关键性空缺。当前文献过度聚焦于分类模型准确率的优化竞赛，却普遍忽视了对抑郁识别作用机制的系统性探究。

中文领域的研究困境更为显著：首先，经临床验证的开源中文数据集严重匮乏，现有数据多源于社交媒体爬取，存在标注粗糙、缺乏病程信息等问题；其次，针对汉语特性的研究尤为稀缺，且缺乏对中文语法特性与抑郁表征关系的深入探讨；此外，医疗文本数据多通过问诊语音数据转换得到，不能保留完整的语言特征，制约了基于访谈文本的抑郁识别研究。

1.3 主要工作

1.3.1 研究目的

根据1.2.4节中描述的挑战，如何解决以上问题成为了本文关注的重点任务。针对以上挑战，本文需要从中文抑郁识别任务中探索语义单元与数据标签的内在映射联系，获取数据集和设计抑郁识别任务便成为了本文的主要研究目的。而医疗数据的收集涉及到隐私问题与资源消耗，如何在有限的时间内获得高质量的数据是个难点。由于特征工程是模型学习的输入基础，模型架构决定特征到决策的映射效率，数据选用约束模型的优化目标，因此如何确定有效的文本特征，如何构建高性能的模型结构，如何选用访谈问题数据便也成为了抑郁识别实验所需要考虑的因素。此外，如何从众多的实验变量中筛选出合适的基线

变量，如何评估因素间可能存在的相互作用对识别效果的影响则是在设计任务中需要考虑的细节问题。最后，如何将实验中的参数进行高效调优以提升抑郁识别性能，如何提供可解释的分类决策依据也是本文所关注的。

1.3.2 研究内容

针对上述目的，本文开展了系统的研究，主要工作如下：

(1) 数据构建方面，本文选用了兰州大学合作医院采集的标准化临床访谈录音，经自动语音识别与专业人工标注，构建了一个经临床验证的中文抑郁访谈文本数据集 DepInter-CN。

(2) 任务设计方面，本文从特征工程、模型架构、数据选用三个方面构建影响因素分析框架，在此之前本文通过分别选用离散式表示与分布式表示的代表性方法构建文本特征空间并使用一系列机器学习模型进行预实验，筛选出可用于后续实验的基线变量。同时考虑到因素间存在的相互影响，通过排除法逐步排除较差表现的自变量。

(3) 特征机理方面，本文通过词袋模型与预训练语言模型的识别性能对比实验，揭示了中文语义与文本特征空间的适配情况。

(4) 模型优化方面，本文设计了一系列深度学习模块组合实验，评估各模型与特征的融合效果，最终提出了一种混合架构 DepHybrid，达到了 77% 的准确率及 F1 分数。

(5) 数据选用方面，本文通过解构访谈问题的语义组合，分析了各类问题得到的情感线索，发现情况问诊的效果整体上要优于图片描述，负性和中性问诊问题的效果优于正性问题。

(6) 参数调优方面，本文引入了多臂老虎机理论，设计基于 UCB 的超参数自适应优化器，使准确率提高了 0.23%，召回率提升了近 10.3%，AUC 提升了近 4.5%。

(7) 可解释性方面，本文使用了 t-SNE 与 SHAP 方法进行可视化特征分布及特征重要程度，验证了模型架构的可信度。

1.3.3 主要贡献

本文针对中文抑郁识别研究中的数据缺陷、特征机理模糊、模型适应性不足等核心问题，在理论与方法层面取得以下创新性突破：

(1) 构建了经临床验证的中文抑郁访谈数据集 DepInter-CN。本文使用了通过标准化访谈协议采集的三甲医院语音数据，并进行专家转写与标注，突破现有社交媒体数据标注粗糙的局限，为中文抑郁识别研究提供高质量基准数据集。

(2) 提出了混合架构 DepHybrid 与自适应优化范式。本文设计了基于预训练模型的双分支集成架构，融合了机器学习与深度学习方法，取得了良好的抑郁识别性能；通过 UCB 算法实现了超参数动态优化，以 LR 模型进行测试，在同等计算资源下使准确率提高了 0.23%，召回率提升了近 10.3%，AUC 提升了近 4.5%（对比单模型基准）。

(3) 揭示了中文语义的特征空间适配场景。本文通过对比词袋模型与预训练模型的特征空间映射性能，发现 ERNIE 实体级别的掩码（ELM）和知识图谱的融合能更好地理解文本与特征的语义关联；通过解构访谈问题的语义组合，分析各类问题得到的情感线索，发现情况问诊的效果整体上要优于图片描述，负性和中性问诊问题的效果优于正性问题，从而为特征工程提供依据。

(4) 验证了抑郁识别架构的可解释性。本文通过使用 t-SNE 与 SHAP 方法，提高了人工智能技术在医疗场景中的透明度，展示出模型的特征分布情况，为医疗场景中可解释人工智能的分析提供了研究范式。

1.4 文章结构安排

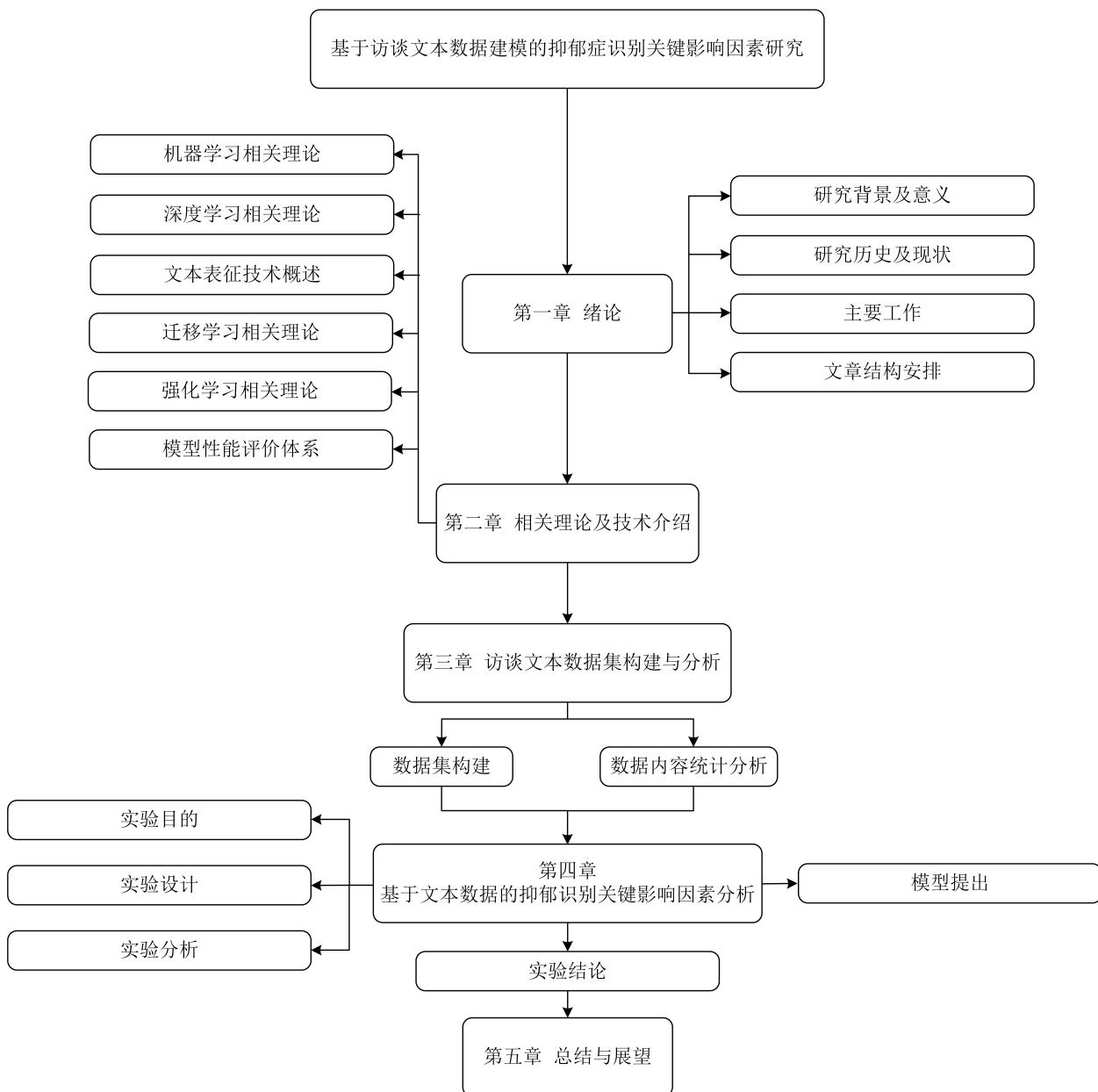


图 1.3 本文的结构框架

本研究采用递进式研究架构（如图1.3所示），遵循“理论奠基→数据构建→方法创新”的闭环研究范式，形成有机衔接的五章论述体系：

第一章系统阐释了抑郁症的社会危害与研究价值，揭示领域研究发展情况，并基于研究空白确立技术路线。

第二章介绍了理论基础及常用技术，重点解析文本挖掘的特征工程方法、机器学习模型与深度学习框架的核心机理。

第三章呈现了数据集构建流程，详细说明从兰州大学合作医院采集的标准化语音数据转写流程，以及基于 PHQ-9 及 M.I.N.I. 诊断标准的标注情况。

第四章为核心方法论创新章节，通过设计控制变量实验量化评估特征构造策略、模型架构选择、数据内容选用对算法性能的贡献情况。

第五章总结研究成果，同时从多模态数据融合和个性化诊断模型等维度规划后续研究方向。

1.5 本章小结

本章主要对文本数据抑郁识别领域的相关研究进行了系统性综述，为后续研究提供了理论框架与分析基础。本章分为四个主要部分：首先是研究背景及意义，阐述了抑郁症的危害与研究价值；之后进行文献发展脉络梳理，按时间顺序归纳了抑郁识别研究从早期基于心理学量表的规则分析、中期传统机器学习方法的应用，到近年来深度学习与预训练模型主导的阶段特征；然后是现存挑战与不足剖析，重点讨论了数据质量、特征提取、模型构建、问题范式等关键瓶颈问题；最后是研究思路与结构规划，明确了本研究将通过关键影响因素分析突破现有局限，简要说明了后续章节的结构安排。

第二章 相关理论及技术介绍

2.1 引言

随着人工智能技术的快速发展，机器学习作为数据驱动方法的核心范式，为复杂模式识别与预测任务提供了理论基础。近年来，深度学习通过构建多层次非线性变换架构，在计算机视觉、自然语言处理等领域展现出超越传统方法的表征学习能力。本文聚焦于文本情感分析任务，该领域需要融合语言特征表示与上下文理解能力，同时对模型的跨领域适应性和决策优化机制提出了更高要求。本章将系统阐述支撑本研究的理论体系与技术方法，涵盖机器学习基础理论、深度学习模型演进、文本特征工程技术、迁移学习与强化学习范式，以及模型评估指标体系。

首先，本章将从机器学习基础理论切入，解析常见模型的工作原理，为后续深度学习模型的构建奠定数学基础。继而重点探讨深度学习理论框架，包括卷积神经网络的空间特征提取机制、循环神经网络对序列数据的建模优势，以及注意力架构在长距离依赖建模中的突破性进展。在文本特征技术层面，将系统梳理从传统 TF-IDF 统计表征到动态上下文嵌入的技术演进。

针对情感分析任务中的领域适应性挑战，本章将阐释迁移学习的核心理论，解析特征迁移与模型微调策略在跨领域知识迁移中的应用路径。同时，为提升模型在交互式场景中的决策能力，引入强化学习的价值函数优化框架，探讨其在动态策略调整与长期回报优化中的独特价值。最后，本章将构建完整的模型评估体系，详细解析准确率、召回率、F1 值等分类指标的计算逻辑，并讨论交叉验证、对抗测试等评估方法在模型鲁棒性验证中的应用规范。通过多维度的理论铺垫与技术剖析，为后续创新性模型的提出奠定坚实基础。

2.2 机器学习相关理论

2.2.1 统计学习方法

1. 逻辑回归

逻辑回归（LR）作为广义线性模型的典型代表，通过 Sigmoid 函数将线性组合映射为概率，满足 $p(y=1|x) = \frac{1}{1+\exp(-(w^\top x+b))}$ 。该模型源于概率响应理论，兼具可解释性和计算效率，在因果推断领域应用广泛。其参数通过最小化带 L2 正则化的负对数似然函数进行估计：

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^N [y_i \ln p_i + (1-y_i) \ln(1-p_i)] + \lambda \|w\|_2^2 \quad (2.1)$$

尽管在非线性可分数据和高维稀疏场景中存在局限，但可通过集成学习或深度学习方法进行增强。模型核心优势在于概率输出的严格数学定义和清晰的线性决策边界。

2. 支持向量机

支持向量机 (SVM) 是一种基于结构风险最小化原则的分类方法，通过最大化间隔超平面实现分类^[91]，其核心思想是将低维线性不可分数据通过核函数（如高斯核、多项式核）隐式映射至高维特征空间以解决非线性问题^[92]。针对线性可分数据，SVM 通过最小化权重范数

$$\min \frac{1}{2} \|w\|^2 \quad (2.2)$$

并满足约束 $y_i(w^\top x_i + b) \geq 1$ 获得最优分离超平面；对于含噪声或非线性数据，引入松弛变量 ξ_i 和惩罚系数 C 构造软间隔模型

$$\min \frac{1}{2} \|w\|^2 + C \sum \xi_i \quad (2.3)$$

以提升鲁棒性。通过拉格朗日对偶转换，问题转化为

$$\max_{\alpha} \sum \alpha_i - \frac{1}{2} \sum \alpha_i \alpha_j y_i y_j K(x_i, x_j) \quad (2.4)$$

约束 $0 \leq \alpha_i \leq C$ 且 $\sum \alpha_i y_i = 0$ ，最终仅支持向量 ($\alpha_i > 0$) 参与构建决策函数

$$f(x) = \text{sign} \left(\sum_{SV} \alpha_i y_i K(x_i, x) + b \right) \quad (2.5)$$

SVM 在小样本高维数据中表现优异，但其 $O(n^3)$ 的计算复杂度限制了大规模应用，需依赖 SMO 算法加速和交叉验证优化参数 (C, γ)，虽具有全局最优解和强泛化能力，但对噪声敏感且核函数选择依赖经验，需结合实际场景权衡设计。

3. k 近邻算法

k 近邻 (k -NN) 是一种基于局部相似性假设的非参数监督学习方法，其核心思想认为特征空间中邻近样本具有相似性。该方法通过计算目标样本与训练集的距离（如欧氏距离、曼哈顿距离或面向文本的余弦相似度），按距离排序选取前 k 个近邻（ k 值常通过交叉验证或经验公式 $k = \lfloor \sqrt{N} \rfloor$ 确定），进而进行分类或回归预测：分类任务采用多数表决机制，以最高票类为结果；回归任务则通过逆距离加权（权重 $w_i = 1/(d + \epsilon)$ ）融合近邻输出，避免零除误差。由于决策过程透明直观，该方法在医学诊断、金融风险评估等需高可解释性的领域具有显著优势。

4. 朴素贝叶斯

朴素贝叶斯分类器 (NBC) 基于贝叶斯决策理论与特征条件独立性假设，通过后验概率最大化实现分类，其核心公式为

$$P(Y = c_k | \mathbf{X}) \propto P(Y = c_k) \prod_{i=1}^n P(X_i | Y = c_k) \quad (2.6)$$

其中先验概率 $P(Y = c_k)$ 采用平滑极大似然估计 ($\alpha = 1$)，离散特征条件概率引入拉普拉斯平滑 ($\lambda = 1$)，连续特征则假设服从高斯分布。该模型通过条件独立性假设将联合概率分解

为特征边际概率乘积，虽可能弱化现实数据关联性，但显著降低了计算复杂度至 $O(n|\mathcal{C}|)$ ，尤其适用于文本分类、医疗诊断等高维稀疏场景。预测时通过

$$\hat{y} = \operatorname{argmax}_{c_k} \left[\log P(Y = c_k) + \sum_{i=1}^n \log P(X_i | Y = c_k) \right] \quad (2.7)$$

避免数值下溢，直接输出最大对数后验概率对应的类别。

5. 决策树学习

决策树（DT）作为非参数监督学习方法，其理论演进历经 ID3、C4.5 和 CART 三大里程碑，通过递归二分策略构建可解释性强的树状结构（时间复杂度 $O(nm \log m)$ ），广泛应用于医疗诊断、信用评估等场景。核心分裂准则随算法迭代优化：ID3 率先采用信息增益

$$IG(D, A) = H(D) - \sum_v \frac{|D_v|}{|D|} H(D_v) \quad (2.8)$$

但存在多值特征偏好；C4.5 引入信息增益率

$$IGR(D, A) = IG(D, A) / IV(A) \quad (2.9)$$

通过固有值 $IV(A)$ 抑制特征分裂偏差；CART 创新性地使用基尼不纯度

$$Gini(D) = 1 - \sum p_k^2 \quad (2.10)$$

进行分类，回归任务则通过最小化均方误差

$$MSE = \frac{1}{N} \sum (y_i - \hat{y})^2 \quad (2.11)$$

实现。树生长终止条件涵盖节点样本量阈值 ($\tau_{min} = 10$)、纯度阈值 (Gini<0.01)、MSE 变化量 ($\delta = 0.005$) 和特征耗尽四重约束，辅以预剪枝（最大深度 $d_{max} = 8$ 、最小增益 $\gamma = 0.05$ ）和后剪枝（代价复杂度系数 $\alpha = \frac{R(t) - R(T_t)}{|L(T_t)| - 1}$ ）双重策略控制过拟合。三大算法特征鲜明：ID3 仅支持离散特征选择，C4.5 拓展至连续特征并改进分裂准则，CART 则以二叉树结构统一处理分类与回归任务，形成完整的决策树方法论体系。

6. 高斯过程

高斯过程（GP）作为一种非参数贝叶斯模型，通过核函数定义函数空间的先验分布，对任意有限样本集 $\{\mathbf{x}_i\}_{i=1}^n$ ，其隐函数值 \mathbf{f} 服从零均值的联合高斯分布 $\mathcal{N}(\mathbf{0}, \mathbf{K})$ ，其中协方差矩阵 \mathbf{K} 由核函数 $k(\mathbf{x}_i, \mathbf{x}_j)$ 构造。在回归任务中，含高斯噪声 $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ 的观测数据 \mathcal{D} 可通过解析解获得预测分布 $\mathcal{N}(\mu_*, \sigma_*^2)$ ，其中均值和方差分别为

$$\mu_* = \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y} \quad (2.12)$$

$$\sigma_*^2 = k_{**} - \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_* \quad (2.13)$$

采用径向基核 (RBF) $k_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp(-\|\mathbf{x} - \mathbf{x}'\|^2/(2l^2))$ 时, 超参数 $\theta = (\sigma_f, l, \sigma_n)$ 通过最大化对数边际似然 $\log p(\mathbf{y}|\mathbf{X}, \theta)$ 优化。对于二分类任务, 引入隐变量 $f(\mathbf{x}) \sim \mathcal{GP}$ 并通过 sigmoid 函数映射 $p(y=1|\mathbf{x}) = \sigma(f(\mathbf{x}))$, 后验推断采用拉普拉斯近似: 先求解后验众数 $\hat{\mathbf{f}}$, 再以高斯分布 $\mathcal{N}(\hat{\mathbf{f}}, (\nabla \nabla \log p(\mathbf{f}|\mathcal{D})|_{\hat{\mathbf{f}}})^{-1})$ 近似后验, 预测时通过数值积分计算类别概率

$$p(y_* = 1|\mathbf{x}_*, \mathcal{D}) \approx \int \sigma(f_*) q(f_*|\mathcal{D}) df_* \quad (2.14)$$

2.2.2 在线学习方法

1. 被动攻击学习算法

被动攻击算法 (PAA) 是一种高效在线学习方法, 专为处理非平稳数据流 (如实时文本分类) 设计。其核心思想是通过动态平衡模型稳定性与适应性, 以单样本更新时间复杂度 $O(d)$ (d 为特征维度) 实现高效更新, 显著优于传统在线 SVM 的 $O(d^2)$ 。算法对时刻 t 的样本 (\mathbf{x}_t, y_t) 依次执行四步操作: 首先计算分类置信度 $f_t = y_t \mathbf{w}_t^\top \mathbf{x}_t$, 若 $f_t \geq 1$ 则保持模型 \mathbf{w}_t 不变 (被动模式); 否则基于间隔损失 $\ell_t = \max(0, 1 - f_t)$ 触发攻击模式, 通过自适应步长 $\eta_t = \ell_t / (\|\mathbf{x}_t\|_2^2 + \frac{1}{2C})$ (其中正则化参数 $C > 0$ 控制模型复杂度与拟合能力的平衡) 执行梯度投影更新 $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t y_t \mathbf{x}_t$ 。该过程等价于求解约束优化问题

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w} - \mathbf{w}_t\|_2^2 \text{ s.t. } y_t \mathbf{w}^\top \mathbf{x}_t \geq 1 \quad (2.15)$$

在保证更新效率的同时维持模型鲁棒性。

2.2.3 集成学习方法

1. 堆叠泛化

堆叠泛化 (Stacking) 作为集成学习的经典范式 (图2.1), 通过基模型与元模型的双层协同优化提升预测性能。其核心流程首先基于 SVM、随机森林等异构算法训练多样化基模型集合, 随后通过 N 折交叉验证策略规避数据泄露: 将每个基模型在验证集上的预测结果横向拼接为元特征矩阵, 最终以该矩阵为输入特征、原始标签为目标, 训练具有强解释性的元模型 (如线性回归或 XGBoost), 形成复合预测函数 $\hat{y} = \mathcal{M}_{meta}([\mathcal{M}_1(\mathbf{x}), \dots, \mathcal{M}_K(\mathbf{x})])$ 。该架构通过基模型的局部泛化与元模型的全局整合, 实现多层次特征表达的深度融合。

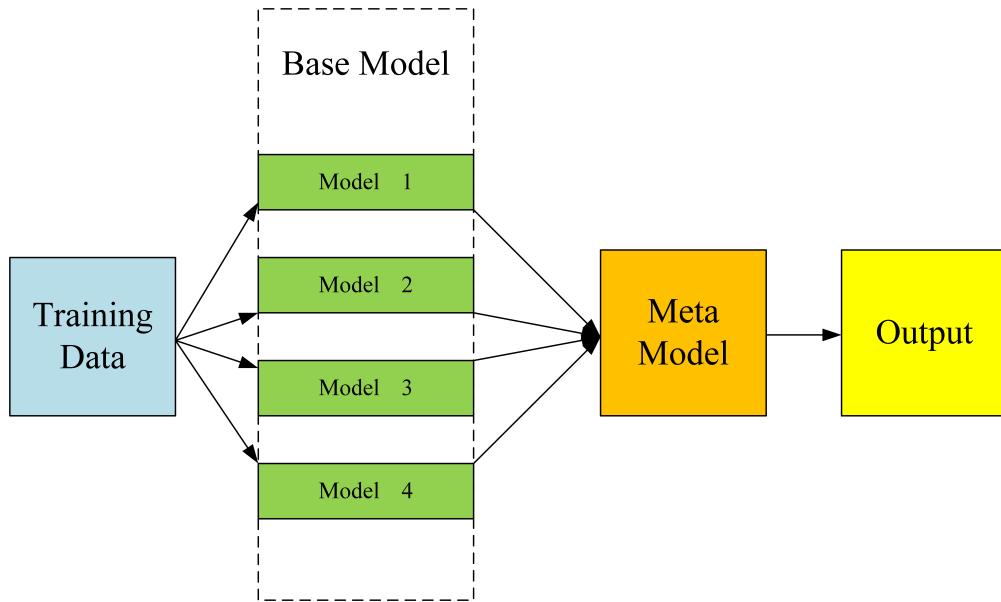


图 2.1 Stacking 方法

2. 袋装算法

袋装算法 (Bagging) 是一种通过自助采样构建多样性基学习器集合的集成方法 (图2.2)，旨在降低模型方差，尤其适用于提升高方差模型 (如决策树) 的稳定性。其核心流程包括：首先对训练集 $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ 进行 T 轮 (通常 $T = 100$) 有放回抽样，生成容量为 n 的子集 \mathcal{D}_t ，其中单样本每轮未被采样的概率为 $(1 - \frac{1}{n})^n \approx 36.8\%$ ，因此每个子集约含 63.2% 原始数据，剩余 36.8% 作为袋外 (OOB) 样本用于误差估计；随后基于各子集独立训练同质基模型 (如决策树)；最终通过多数投票 (分类任务， $\hat{y} = \operatorname{argmax}_c \sum_{t=1}^T \mathbb{I}(\mathcal{M}_t(\mathbf{x}) = c)$) 或均值聚合 (回归任务， $\hat{y} = \frac{1}{T} \sum_{t=1}^T \mathcal{M}_t(\mathbf{x})$) 完成预测集成。

随机森林 (RF) 与极端随机树 (Extra tree) 均为基于 Bagging 的集成学习算法，核心差异在于随机化策略与节点分裂机制。随机森林通过 Bootstrap 采样生成训练子集 (保留约 63.2% 原始样本)，并在决策树节点分裂时从全部特征中随机选择 $m = \sqrt{d}$ 个候选特征^[93]，基于基尼指数或信息增益搜索全局最优分裂点，通过双重随机性降低过拟合^[94]。极端随机树则进一步强化随机性，允许直接使用全部特征 ($m = d$)，并在每个候选特征上随机生成 k 个分裂阈值 (默认 $k = 10$)，仅选择局部最优阈值而非全局搜索，使时间复杂度从 $O(mn \log n)$ 降至 $O(mk)$ 。该机制显著降低模型方差，更适用于高噪声数据场景。

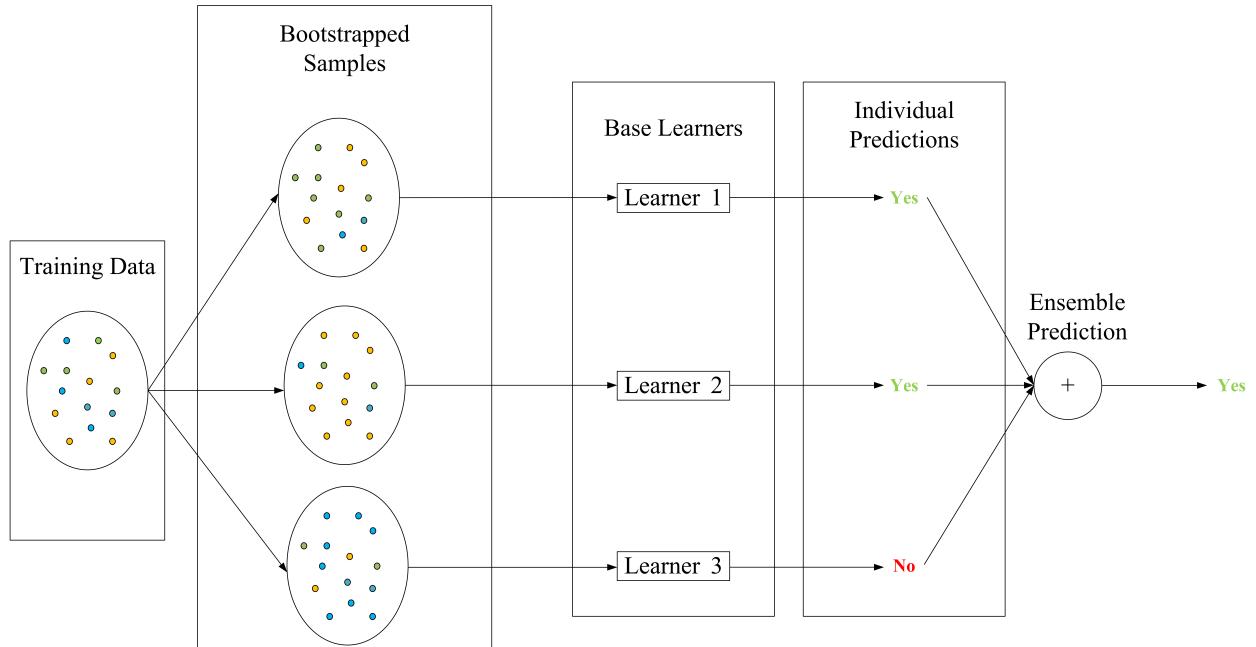


图 2.2 Bagging 方法

3. 提升方法

提升算法 (Boosting) 通过序贯训练弱学习器动态调整样本权重，聚焦误分类样本以降低模型偏差，最终通过加权多数表决构建强预测模型 (图2.3)。其核心流程为：初始化样本权重为均匀分布后，在每轮迭代中基于当前权重训练弱学习器，计算模型权重 $\alpha_t = \frac{1}{2} \ln((1 - \epsilon_t)/\epsilon_t)$ (ϵ_t 为加权误差)，并通过指数损失函数更新样本权重 $w_i^{(t+1)} \propto w_i^{(t)} \exp(-\alpha_t y_i \mathcal{M}_t(\mathbf{x}_i))$ ，经归一化后传递至下一轮训练。最终组合模型 $\mathcal{M}(\mathbf{x}) = \text{sign}(\sum_{t=1}^T \alpha_t \mathcal{M}_t(\mathbf{x}))$ 通过强化有效弱分类器的作用，显著提升高偏差场景下的预测性能。

Boosting 算法通过迭代集成弱学习器提升模型性能，其中六种经典算法各具特色：AdaBoost 作为开创性算法，通过动态调整样本权重与指数损失函数 $\mathcal{L} = \sum e^{-y_i F(x_i)}$ 构建弱分类器组合，虽理论严谨但对噪声敏感；随后提出的 GBDT 以梯度下降为核心，通过拟合伪残差 $r_{mi} = y_i - F_{m-1}(x_i)$ 迭代生成 CART 树，并在叶节点输出残差均值 $\gamma_{jm} = \sum r_{mi} / |R_{jm}|$ ；XGBoost 进一步引入正则化项 $(\gamma T + \lambda \|w\|^2)$ 与二阶泰勒展开，推导出叶节点权重解析解 $w_j^* = -\frac{\sum g_i}{\sum h_i + \lambda}$ ，平衡精度与复杂度；LightGBM 针对大规模数据优化，采用特征离散化 (255 bins)、梯度采样 (Top20%+ 随机 5%) 和特征捆绑策略，显著提升计算效率；HGB 则通过加权分箱预算梯度统计量 G_{ik}, H_{ik} ，并基于分裂增益公式 $\text{Gain} = \frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{(G_L + G_R)^2}{H_L + H_R}$ 加速决策树生长；专攻类别特征的 CatBoost 通过动态排列样本与对称平均 $\hat{y}_i = \frac{1}{2}(f(x_i|\pi) + f(x_i|\pi'))$ 解决预测偏移问题，并内置类别特征处理机制。这些算法均基于梯度优化框架，差异主要体现在特征离散化策略 (如直方图分箱)、正则化设计 (XGBoost 的结构风险) 及工程实现优化 (LightGBM 的特征捆绑)，共同推动集成学习在效率和泛化能力上的突破。

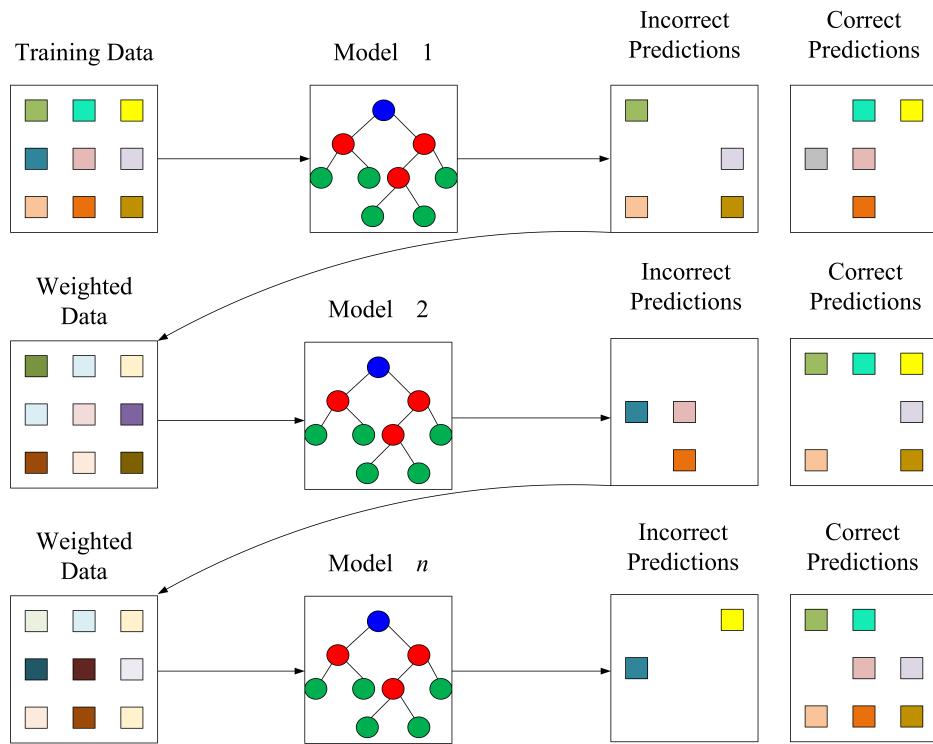


图 2.3 Boosting 方法

2.2.4 自动机器学习

自动机器学习 (AutoML) 通过三级协同优化框架重构传统建模流程：在流程自动化层面，元学习（如 Auto-Sklearn）通过元特征空间映射跨数据集泛化预测最优算法组合，结合遗传编程（如 TPOT）以多目标函数进化建模管道，平衡模型性能与计算复杂度；架构搜索领域，DARTS 创新性地将离散拓扑选择松弛为连续可微分优化问题，通过双层交替优化（架构参数 α 与网络权重 w ）实现高效搜索，并引入正则化约束防止架构退化；超参优化中，贝叶斯优化基于高斯过程与 Matérn 核建模参数空间响应曲面，而 Hyperband 通过动态资源分配策略（逐轮淘汰与几何级数预算衰减）提升搜索效率。三级优化体系从算法选择、架构生成到参数调优形成闭环，共同实现机器学习全流程的智能自动化。

2.3 深度学习相关理论

人工神经网络 (ANN) 的发展历程历经三大里程碑：1943 年 McCulloch-Pitts 神经元模型奠定计算基础，1986 年反向传播算法突破参数优化瓶颈，2012 年后深度网络推动 AI 革命。其架构通过仿生学原理构建：输入层维度与数据特征严格匹配（如图像输入 $d_{in} = H \times W \times C$ ）；隐藏层通过 $\mathbf{h}^{(l)} = \text{ReLU}(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$ 实现非线性特征变换；输出层则根据任务需求采用 Softmax 分类或线性回归。训练过程依托反向传播的链式求导机制，结合优化算法演进——从基础 SGD、引入动量加速的 Momentum，到自适应矩估计的 Adam（默认超参数 $\beta_1 = 0.9, \beta_2 = 0.999$ ），形成高效参数更新范式。为防止过拟合，L2 正则化约束权重幅值、

Dropout 随机失活神经元 ($p = 0.5$ 等效集成 2^N 子网络)、早停机制 (验证集损失连续 10 轮未降则终止训练) 等正则化技术协同提升模型泛化能力，共同构成现代 ANN 的核心训练框架。

2.3.1 前馈神经网络

前馈神经网络 (FNN) 是一种单向传播的人工神经网络模型，由输入层、隐藏层和输出层构成，层间全连接而层内无连接（图2.4）。其通过多层非线性变换逼近复杂函数关系，数学表达为：

$$\mathbf{y} = f^{(L)}(\mathbf{W}^{(L)}f^{(L-1)}(\cdots f^{(1)}(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(L)})$$

其中每层包含权重矩阵 \mathbf{W} 、偏置 \mathbf{b} 和非线性激活函数 f ，信息严格单向传递（输入 \rightarrow 隐藏层 \rightarrow 输出）。当层数超过 2 时称为深度神经网络 (DNN) 或全连接网络 (FCNN)，2 层结构则称为多层次感知机 (MLP)。根据 Cybenko 定理，仅需单隐藏层且采用 Sigmoid 类激活函数，即可在紧致集上以任意精度逼近连续函数，体现其通用近似能力。网络参数（权重与偏置）通过机器学习自动优化，实现对输入信号的非线性复合变换与高维特征提取。

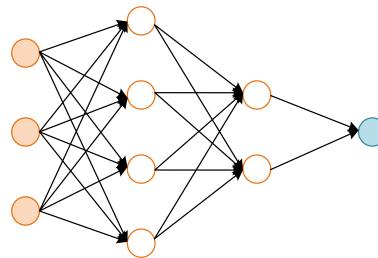


图 2.4 前馈神经网络

卷积神经网络 (CNN)^[95] 通过局部感受野、权值共享和下采样机制（图2.5）高效处理图像等网格数据，其参数量仅需 $O(k^2C)$ (k 为卷积核尺寸， C 为通道数)，有效缓解高维数据过拟合问题。其分层架构包含卷积层（局部加权生成特征图）、ReLU 激活函数、池化层（最大/平均池化压缩特征）和全连接层（Softmax 分类），实现从边缘特征到语义特征的渐进抽象。在此基础上，文本卷积神经网络 (TextCNN)^[96] 将文本映射为词向量矩阵 $E \in \mathbb{R}^{n \times d}$ ，采用多尺度卷积核 ($h = 2, 3, 4$) 提取局部上下文特征，经最大池化筛选显著特征后拼接不同尺度的抽象表示，最终通过全连接层输出分类概率。该模型通过空间维度卷积操作，将 CNN 在图像处理中的层次化特征提取能力迁移至文本语义建模领域。

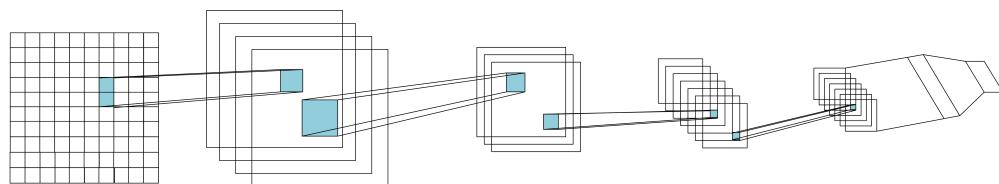


图 2.5 卷积神经网络

2.3.2 反馈神经网络

反馈神经网络 (RNN, 即循环神经网络) 是一种专为序列数据处理设计的动态网络结构, 其核心在于通过循环连接实现时序信息的传递与记忆 (图2.6)。网络在每个时间步共享参数, 利用隐层状态 h_t 动态捕获序列的上下文依赖关系: 其状态更新遵循 $h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$, 输出则由 $y_t = W_{hy}h_t + b_y$ 生成, 其中 σ 为激活函数。RNN 通过持续迭代的隐层状态建模历史信息, 能够有效处理时间序列预测、语言模型等需上下文关联的任务, 与前馈神经网络的单向传播不同, 其循环特性赋予网络双向信息流动的记忆能力, 从而实现对时序动态的精准刻画。

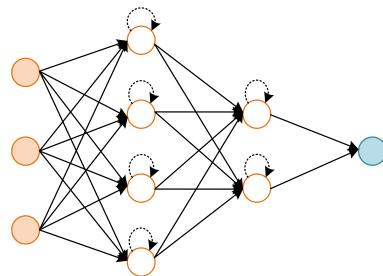


图 2.6 反馈神经网络

长短期记忆网络 (LSTM)^[97] 通过遗忘门、输入门、输出门与细胞状态的协同机制 (图2.7), 有效解决 RNN 梯度消失问题: 遗忘门通过 $f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$ 控制历史信息保留比例, 输入门利用 i_t 和 \tilde{C}_t 生成候选状态, 并通过 $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$ 融合新旧信息, 最终由输出门 o_t 调节隐状态 $h_t = o_t \odot \tanh(C_t)$ 输出。其扩展版本双向 LSTM (BiLSTM)^[98] 通过前向 \overrightarrow{h}_t 与反向 \overleftarrow{h}_t 时序建模, 以特征拼接 $h_t^{bi} = [\overrightarrow{h}_t \parallel \overleftarrow{h}_t]$ 同时捕获上下文依赖, 显著提升序列建模能力。该架构通过门控机制与双向信息流的结合, 实现对长程时序特征的动态筛选与双向感知。

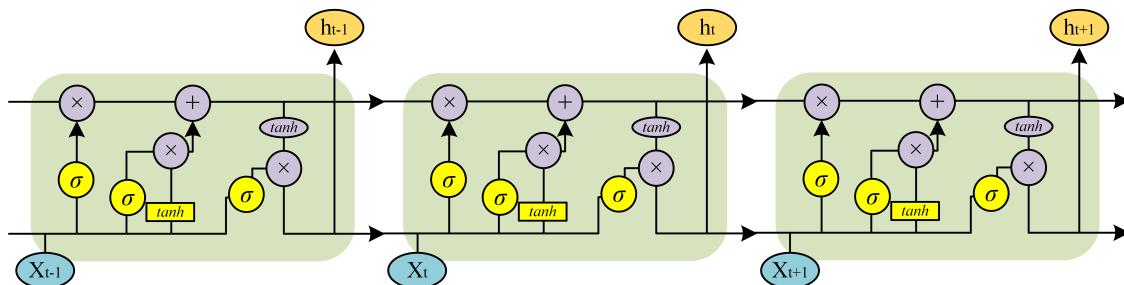


图 2.7 长短期记忆网络

2.3.3 序列到序列模型

序列到序列 (Seq2Seq) 模型基于编码器-解码器框架实现变长序列的端到端转换: 编码器通过循环神经网络 (RNN) 逐时间步压缩输入序列为最终隐藏状态, 或借助 Transformer

的自注意力机制并行捕获全局特征生成状态序列；解码器则以自回归方式，结合前序生成结果和编码器传递的上下文信息逐步预测输出序列。其核心突破在于注意力机制，通过动态计算解码器当前状态与编码器各时刻状态的相关性权重（Softmax 归一化），生成聚焦关键输入片段的上下文向量，替代传统固定上下文表示，从而显著缓解长序列信息衰减问题，提升模型在机器翻译等任务中对长程依赖和跨序列对齐的建模能力。

变换器模型（Transformer）^[83] 是一种基于自注意力机制的深度学习架构（图2.8），通过并行化计算和全局依赖建模显著提升了 Seq2Seq 任务性能。其核心流程首先生成输入嵌入 $\mathbf{X} \in \mathbb{R}^{n \times d}$ 的查询（Q）、键（K）、值（V）向量： $Q = XW^Q$, $K = XW^K$, $V = XW^V$ ，并通过缩放点积计算注意力权重 $\text{softmax}(QK^\top / \sqrt{d_k})V$ ，动态捕获序列全局依赖。为扩展特征表达能力，多头注意力将自注意力分解为 h 个并行子空间，每个头独立学习不同子空间特征，最终拼接并通过 W^O 线性融合。针对序列顺序建模，模型采用正弦/余弦函数生成位置编码 ($PE_{(pos,2i)} = \sin(pos/10000^{2i/d})$, $PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d})$) 以注入位置信息。注意力层后接两层全连接前馈网络，通过 ReLU 激活函数增强非线性表达能力，最终通过堆叠上述模块实现高效的长距离上下文感知与语义表示。

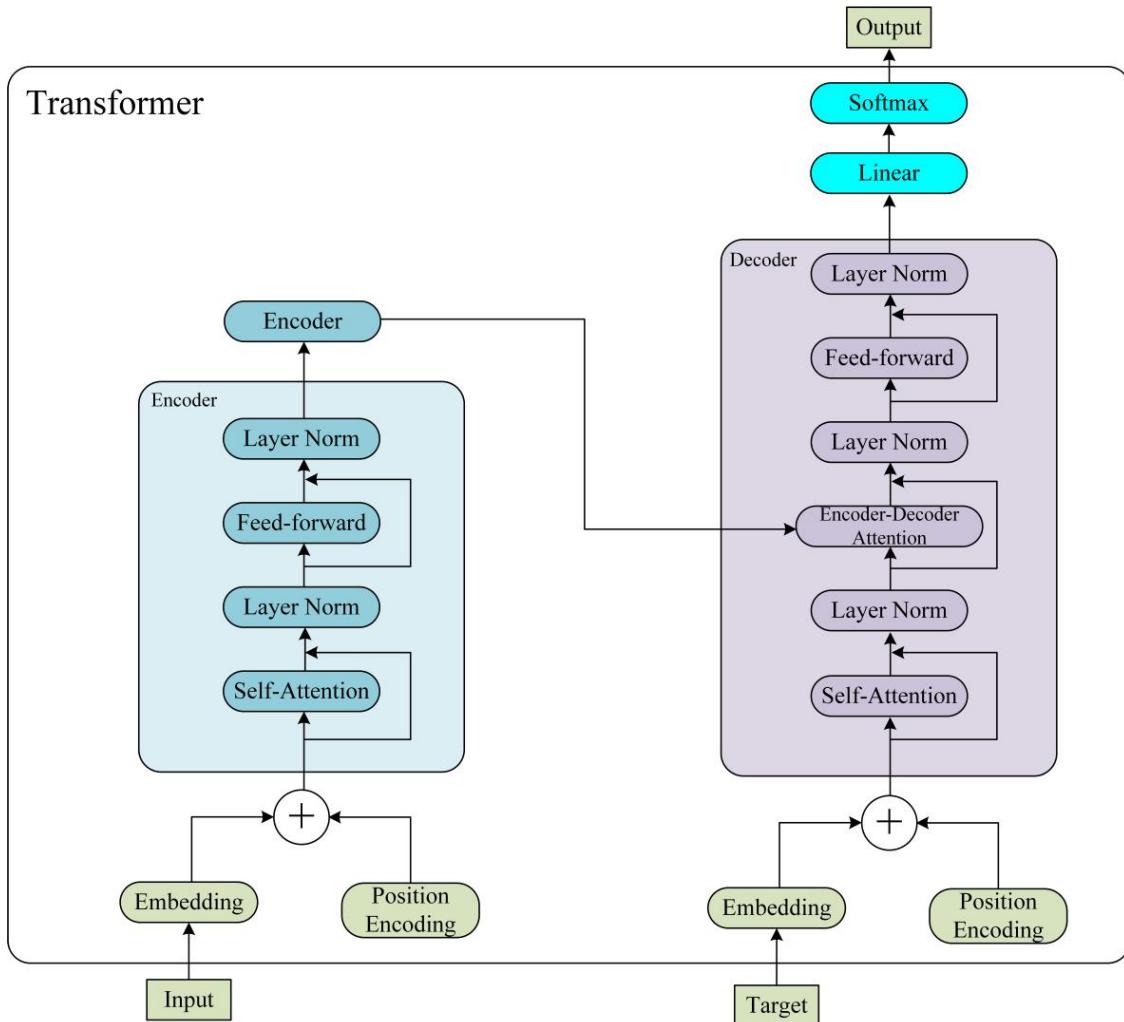


图 2.8 变换器模型

2.3.4 图神经网络

图神经网络 (GNN) 通过消息传递机制处理图结构数据 (图2.9), 在社交网络分析、分子属性预测等领域展现出独特优势。其核心计算范式基于迭代式邻域聚合更新节点嵌入, 数学定义为: $\mathbf{h}_v^{(l+1)} = \sigma(\mathbf{W}^{(l)} \cdot \phi(\{\mathbf{h}_u^{(l)} | u \in \mathcal{N}(v)\}))$, 其中 $\mathcal{N}(v)$ 为节点 v 的邻域 (含自身时引入 self-loop), $\phi(\cdot)$ 为多粒度聚合函数, 具体实现包括均值池化 ($\phi_{\text{mean}} = \sum_u \mathbf{h}_u^{(l)} / |\mathcal{N}(v)|$)、注意力加权 ($\phi_{\text{att}} = \sum_u \alpha_{vu} \mathbf{h}_u^{(l)}$, 权重 α_{vu} 由 MLP 与 softmax 生成) 及图卷积 ($\phi_{\text{gcn}} = \tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} \mathbf{H}^{(l)}$), 其中 \tilde{A} 为带自连接的邻接矩阵, \tilde{D} 为其度矩阵。该机制通过融合局部结构与节点特征, 实现图数据的层次化表征学习。

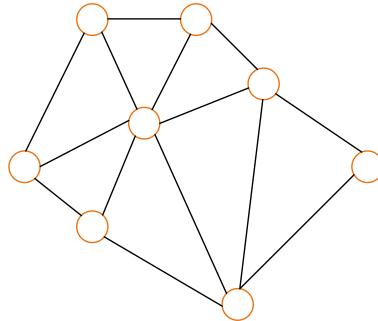


图 2.9 图神经网络

2.4 文本表征技术概述

文本表征技术致力于将自然语言转换为机器可处理的数值形式。传统离散表征方法 (如词袋模型 BoW 和 TF-IDF) 基于词汇统计生成高维稀疏向量, 其计算效率优势与语义表达能力不足形成鲜明对比。伴随深度学习发展, 分布式表征技术 (典型代表包括 Word2Vec 和 GloVe) 通过低维稠密向量实现了语义关联的有效建模。以 BERT 为突破点的预训练模型创新性地运用自注意力机制, 实现了基于上下文环境的动态语义编码。

2.4.1 词袋模型

词袋模型 (BoW) 是一种基于词汇频率统计的文本表示方法, 其核心思想是忽略语法与词序, 将文本视为无序词汇集合。实现过程分为三步: 首先通过分词、停用词过滤、文本清洗及词汇规范化 (如词干提取或词形还原) 完成预处理; 随后构建由语料库唯一词汇组成的维度为 V 的全局词汇表; 最后将文本映射为固定维度的向量, 常用方法包括二值化标记、词频 (TF) 统计或 TF-IDF 加权。其中 TF-IDF 通过逆文档频率 (IDF) 降低高频词权重, 其计算公式为:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log \left(\frac{N}{\text{DF}(t) + 1} \right) \quad (2.16)$$

其中 N 为文档总数, $\text{DF}(t)$ 为包含词项 t 的文档数, $+1$ 项用于平滑未出现词项的零频问题。

2.4.2 词嵌入

词嵌入通过将词汇映射为连续低维向量，克服了传统词袋模型的高维稀疏性与语义缺失问题，其核心是使语义相似的词在向量空间中几何邻近 ($\text{sim}(w_i, w_j) \propto \|\mathbf{w}_i - \mathbf{w}_j\|$)。早期方法如 Word2Vec 采用浅层神经网络，通过 Skip-gram (中心词预测上下文) 和 CBOW (上下文预测中心词) 两种模式分别建模局部语义关联，目标函数分别对应最大化上下文对数概率 $\max \frac{1}{T} \sum_{t=1}^T \sum_{-c \leq j \leq c} \log p(w_{t+j} | w_t)$ 和高效学习高频词表征。GloVe 进一步融合全局共现统计与局部窗口信息，通过带权矩阵分解目标函数 $J = \sum_{i,j} f(X_{ij}) (\mathbf{w}_i^T \tilde{\mathbf{w}}_j - \log X_{ij})^2$ 优化词向量，其中加权函数 $f(X_{ij})$ 抑制高频噪声， X_{ij} 为词对共现频次。随着 Transformer 架构的兴起，以 BERT 为代表的上下文敏感模型通过自注意力机制 $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$ 实现深度双向上下文编码，并借助掩码语言建模 (MLM) 任务随机遮盖 15% 的输入词汇 (80% 替换为 [MASK], 10% 随机替换)，最小化负对数似然 $\mathcal{L}_{\text{MLM}} = -\mathbb{E}_{x \sim D} \sum_{i \in M} \log P(x_i | x_{\setminus M})$ ，联合下一句预测 (NSP) 任务，迫使模型同时捕获一词多义和长程逻辑关联，显著提升了语义建模的深度与灵活性。

2.4.3 中文分词

中文分词 (CWS) 旨在将连续汉字序列转化为符合语义的词语序列，其方法演进可分为两大技术路线。传统方法中，Jieba 分词^[99] 采用词典匹配与统计学习混合架构：通过构建 Trie 树实现基于动态规划的最大概率切分

$$W^* = \arg \max \prod P(w_i | w_{i-1}) \quad (2.17)$$

结合四状态 HMM (B/M/E/S) 处理未登录词，利用维特比算法解码最优序列，并支持多模式切分与领域词典扩展；而 BERT 分词则基于 Transformer 架构，通过 WordPiece 算法迭代合并高频子词

$$(x^*, y^*) = \arg \max \frac{\text{count}(x, y)}{\text{count}(x) \cdot \text{count}(y)} \quad (2.18)$$

以贪心最长匹配实现动态子词切分 ($O(L^2)$ 时间复杂度)，其分词概率与掩码语言模型目标形成闭环，使子词切分与上下文语义嵌入协同优化。两者本质差异在于：Jieba 依赖规则与局部统计特征，适用于术语明确的垂直领域；BERT 则通过全局上下文建模和子词分解，更擅长处理复杂语境与新词涌现，体现了从机械切分到语义驱动的范式跃迁。

2.5 迁移学习相关理论

迁移学习 (TL) 通过迁移源领域的知识提升目标领域模型性能，尤其适用于数据稀缺场景。其核心是利用预训练模型 (如 ImageNet) 的通用特征，通过冻结底层参数保留基础特征提取能力，同时微调顶层参数 (如全连接层) 适配目标任务。这种策略平衡了泛化性与特异性，广泛应用于 CV 和 NLP 领域。实践中需动态调整冻结层位置：过度冻结会限制特征表达，过度微调易导致过拟合，通常通过验证集或学习率策略优化平衡点。

2.6 强化学习相关理论

强化学习 (RL) 以智能体与环境的交互为核心，通过最大化长期累积奖励实现序贯决策优化，其核心挑战在于探索（尝试未知策略）与利用（依赖已知最优策略）的权衡。多臂老虎机 (MAB) 问题作为 RL 的简化模型，假设 K 个独立奖励分布的选项，需在有限次试验中选择最优动作以最大化总收益。为解决这一困境，置信区间上界 (UCB) 算法提出动态平衡机制：通过统计每个动作的历史平均奖励（反映利用价值）和置信区间半径（量化不确定性，驱动探索），以“平均奖励 $+ \alpha \times$ 置信半径”为综合指标选择动作 ($\alpha \geq \sqrt{2}$ 时理论遗憾上界为次线性增长)，实现高效探索-利用平衡。在深度强化学习中，UCB 进一步与神经网络结合形成神经 UCB 框架：通过特征嵌入编码环境状态，结合神经网络预测动作价值，并基于协方差矩阵量化状态-动作不确定度（如 Q 值 = 网络预测 $+ \beta \times$ 特征不确定度），实现端到端的特征学习与自适应探索（通过贝叶斯优化动态调节 β ），在推荐系统、在线广告等动态场景中以高样本效率优化决策，避免过度探索或局部最优陷阱。

2.7 模型性能评价体系

在文本情感分类任务中，构建系统化的评估体系对模型性能进行多维度验证至关重要。本研究采用分类任务标准评估指标与鲁棒性验证方法相结合的评价框架，涵盖基础分类指标、概率分布评估及抗干扰能力测试三个层面。

首先，基于混淆矩阵 (Confusion Matrix) 构建基础分类指标评价体系。如表2.1所示，该矩阵将二分类预测结果划分为真阳性 (TP)、假阳性 (FP)、真阴性 (TN)、假阴性 (FN) 四个象限。核心性能评价体系由以下四个关键指标构成：整体判别准确度 (Accuracy) 体现模型在所有样本中的综合判别能力；正类判别精度 (Precision) 反映预测为正类的样本中真实正类的比例，表征预测结果的可信程度；正类召回率 (Recall) 揭示模型对真实正类样本的识别广度，体现检测出真实正例的全面性；综合平衡指标 (F1 Score) 通过调和平均运算将精度与召回率有机结合，特别适用于正负样本分布不均衡的场景，能更客观地反映模型对少数类别的识别效能^[100]。

表 2.1 混淆矩阵结构示例

真实情绪	预测情绪	
	正向积极	负向消极
正向积极	TP	FN
负向消极	FP	TN

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \quad (2.19)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad (2.20)$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.21)$$

$$\text{F1 Score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.22)$$

其次，引入 AUC-ROC 曲线评估模型在不同决策阈值下的稳定性。通过计算真阳性率 (TPR) 与假阳性率 (FPR) 构建 ROC 曲线，其曲线下面积 (AUC) 量化模型的类别区分能力：

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad (2.23)$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \quad (2.24)$$

$$\text{AUC} = \int_0^1 \text{TPR}(\text{FPR}), d\text{FPR} \quad (2.25)$$

最后，采用交叉验证与对抗测试验证模型鲁棒性。交叉验证通过数据划分稳定性测试防止过拟合，而对抗测试则通过注入噪声数据、领域偏移样本等方式，评估模型在非理想环境下的泛化能力。这种多层次评估体系既能量化模型的分类性能，又可揭示其在实际应用场景中的潜在风险，为模型优化提供明确方向。

2.8 本章小结

本章主要介绍了本研究中涉及的相关理论与技术，为后续的研究内容奠定了基础。本章分为六个主要部分：首先是机器学习相关理论，介绍了机器学习的常用方法；其次是深度学习相关理论，重点阐述了深度学习的核心思想、典型模型；然后是文本表征技术概述，探讨了从传统的基于规则的方法到现代的分布式表征技术的发展历程及其在自然语言处理中的重要性；在迁移学习理论部分，阐释了冻结和微调两种深度神经网络中迁移学习的常用策略；在强化学习部分，介绍了多臂老虎机问题；最后是模型性能评价体系，分析了评估模型性能的常用指标及验证方法。

第三章 访谈文本数据集构建与分析

3.1 引言

在抑郁医疗诊断领域，当前面临着医疗数据专业性高且患者隐私性强等挑战，导致暂时缺乏开源的中文数据集。因此，建立规范、专项的抑郁医疗文本数据集显得尤为重要。本章节的核心任务便是构建这样一个数据集，其数据来源于合作医院提供的医生与患者之间的真实访谈录音。接下来，本章将详尽阐述数据集的采集与标注过程，并对数据内容进行深入的统计分析。同时，为了符合本论文的研究规范，本章还将现实访谈录音数据转化为文字以构建访谈文本数据集。本章节的工作为后续研究提供了坚实的数据基础。

3.2 数据集构建

为了避免重复性数据收集引发的伦理争议和资源浪费，本研究基于刘振宇团队已完成的语音实验数据^[101]，构建了具有医疗场景特征的访谈文本数据集。在数据集设计过程中，我们融合了 Zou 等人提出的半结构化访谈对话方法^[102] 与标准化抑郁评估量表，最终形成包含 9 个主题单元的对话框架：6 段医疗问诊对话（正性、负性各 1 段，中性 4 段）和 3 段图片描述对话（正性、中性、负性各 1 段），具体结构详见表3.1。

表 3.1 访谈文本数据集关键主题内容

主题类别	主题内容
情况问诊对话	旅游计划
	分享一段美好的回忆场景
	最近的情绪状态
	最近的身体状况
	对自己的评价
	描述一件糟糕的经历
图片描述内容	正性表情图片描述
	中性表情图片描述
	负性表情图片描述

图片描述素材选自 CFAPS 情绪图片系统^[103]，该系统包含经本土化评估的 870 张人脸图片，每张图片均通过愉悦度（Valence）、唤醒度（Arousal）和优势度（Dominance）三维度进行标准化标注。本研究从中选取展现高兴（正性）、平静（中性）和悲伤（负性）情绪的三张女性面孔作为视觉刺激材料^[101]。

在数据标注环节，采用 Praat 语音分析软件^[104] 对语音样本进行系统化处理。具体工作流程包括：将 WAV 格式的语音文件^[105] 导入 Praat 平台；人工标注对话角色（医生、患者、家属等）及非语音片段（静默、环境噪声）；转写语音内容为文本；通过 TextGrid 对象保存多层级标注结果。标注过程严格遵循熊子瑜编著的标准化操作指南^[106]，确保标注结果的一致性和可溯性。

3.3 数据内容统计分析

本文使用 50 人完整录音数据作为后文中抑郁情绪分析模型的实验样本数据，其中健康对照 25 名，抑郁患者 25 名。每位受访者数据中均包含 6 条情况问诊以及 3 条图片描述共计 9 条对话信息，受访者样本信息分布情况如表3.2所示。

表 3.2 受访者样本信息分布情况

实验组别	实验样本（人）	情感极性	样本数量	映射关系
健康对照	25	正样本	225	0
抑郁患者	25	负样本	225	1
总计	50	/	450	/

在该项情感分类任务中，医生采用了 M.I.N.I.^[107] 和 PHQ-9 综合判断患者是否患有抑郁症状，并对样本数据进行了映射标注处理。目本文中使用的数据项为 450 条，其中正负样本的比例为 1:1。在完成标注后，访谈文本数据集示例如表3.3所示，文本长度、数据项分布如图3.1所示。

表 3.3 访谈文本数据集示例

样本编号	映射关系	问题编号	主题类别	访谈文本
0	0	4	您最近身体状况如何？对您的生活有哪些影响？	最近身体状况一般，然后正在吃一些中药调理，没有很明显的影响。
1	1	3	您最近情绪如何？这种情绪对您的生活带来了什么影响？	* 正常，没有也没有 * 什么影响。

注：数据项中 * 表示听不清的音

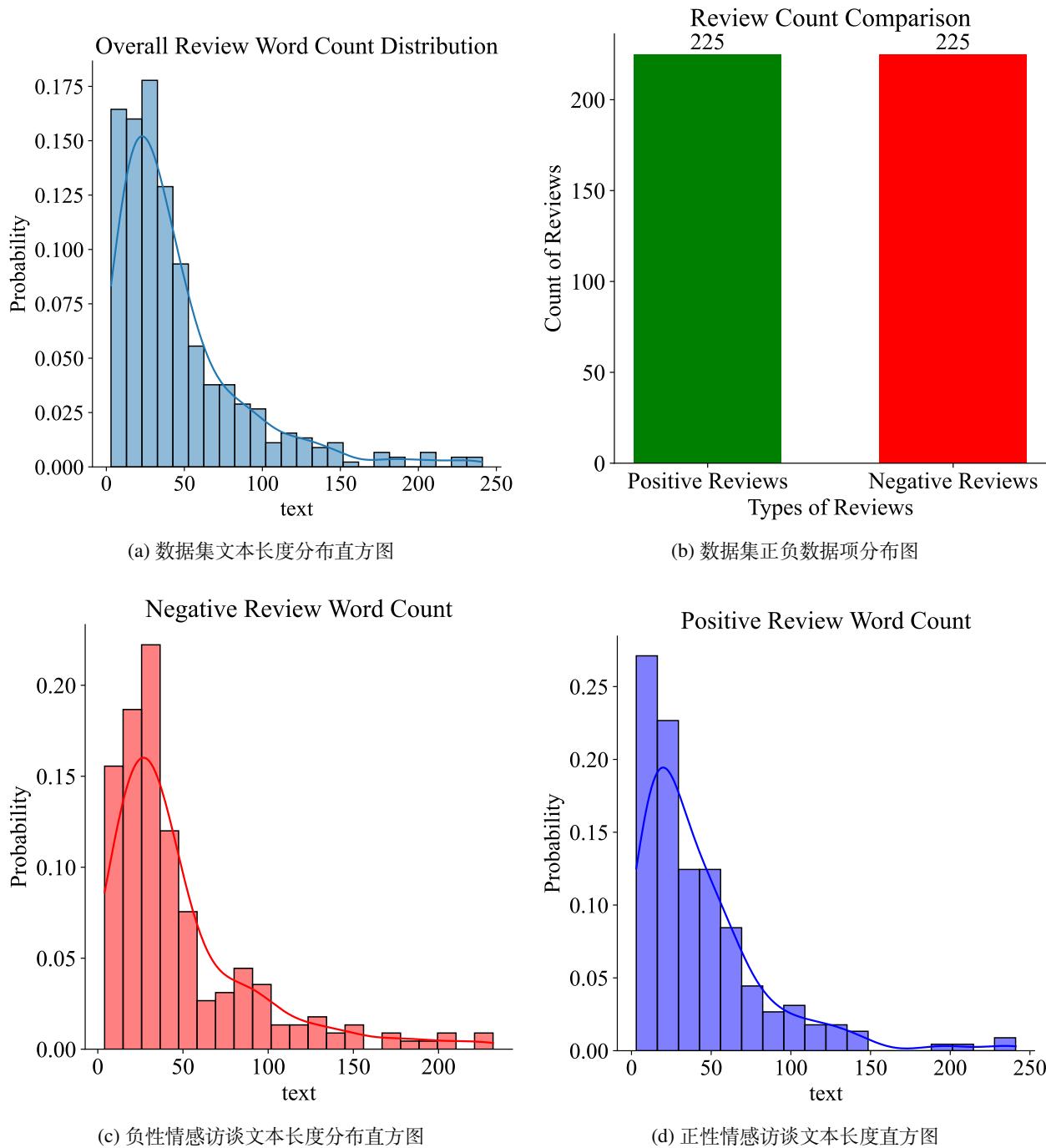


图 3.1 访谈数据集文本长度及数据项分布

通过表3.2及图3.1可以得出，数据集是平衡的，包含相同数量的正负评论语义。大部分的数据项的文本长度均在30个字以内，且最大文本长度不超过250个字，主要集中在150个字以内。

接下来本文通过词云图观察正负情感极性在文本数据中的体现，具体流程包括：使用Jieba工具进行中文分词，并清洗非汉字字符及特殊符号；过滤“的、就是、是”等跨情感类别高频虚词；基于TF-IDF权重生成词云可视化图谱（图3.2）。

词频分析显示，抑郁组第一人称单数“我”的出现频率显著高于对照组，这与 Rude 等人揭示的自我聚焦语言模式^[66]具有一致性。然而，不同情感类别间的显性情感词汇（如“快乐”“痛苦”）分布未呈现统计学显著差异，这增加了基于词汇特征的情感识别难度。值得注意的是，作为语音转写文本，数据集保留了停顿标记、重复性语气词（如“呃”“嗯”）等副语言特征。结合刘振宇团队的声学分析^[101]，抑郁人群的语言行为特征呈现多模态表达特性：在声学维度表现为语速减缓、基频范围收窄；在文本维度表现为话题集中度升高、信息熵降低；而在交叉维度则体现为犹豫标记（如重复修正）的频率增加。这种多模态特性提示，单纯依赖传统分词方法可能丢失关键鉴别特征，需开发能同步捕捉词汇、韵律及语用特征的中文处理框架。

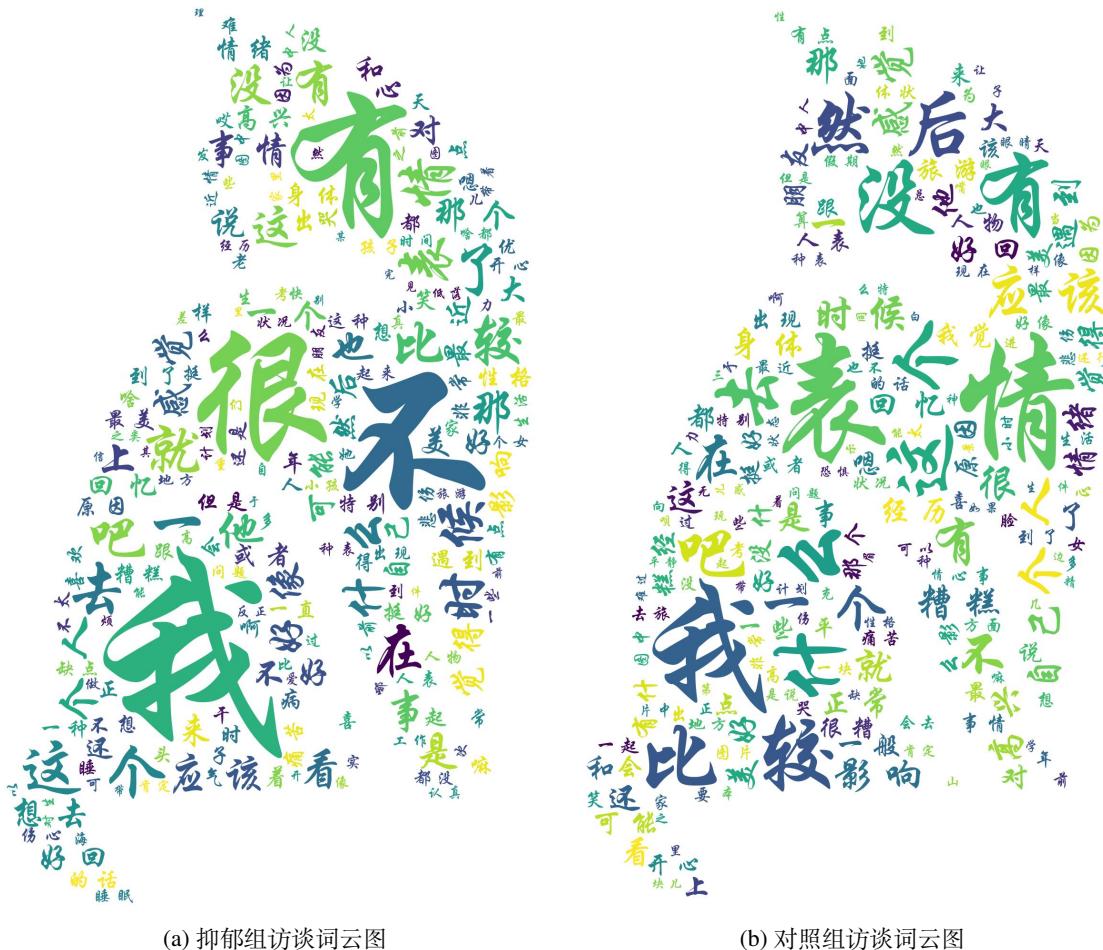


图 3.2 访谈数据集词云图

3.4 本章小结

本章主要介绍了本研究中涉及的访谈文本数据集，为后续的研究内容奠定了基础。本章分为两个主要部分：首先是数据集构建，介绍了数据集的收录及标注工作、主要内容；其次是数据内容统计分析，展示了数据集的样本信息分布情况及数据分布整体概况。

第四章 基于文本数据的抑郁识别关键影响因素分析

4.1 引言

在抑郁情感分析领域，现有研究存在特征表达不充分、模型适配性不足及任务范式待优化等瓶颈，制约着诊断算法的性能提升。本章节的核心任务在于构建抑郁症识别影响因素分析体系，通过设计控制变量实验系统解析特征构造策略、模型结构设计、任务数据选用对诊断效果的贡献。接下来，将详尽阐述控制变量实验的设计原理与实施流程，并对不同变量组合的模型表现进行多维度对比分析。通过构建分层递进的实验验证体系，本章节不仅实证了论文提出模型在抑郁语义特征捕获方面的优越性，也为临床辅助诊断系统研发提供了算法支撑。

4.2 模型提出

4.2.1 DepHybrid 架构

针对中文抑郁识别中特征机理模糊、模型适应性不足的核心问题，本文提出了一种基于 ERNIE 预训练模型的双分支集成神经网络架构 DepHybrid，其核心结构如图4.1所示。该模型通过融合词元级序列特征与句子级全局特征，实现了高效的文本分类任务。以下详细阐述各模块的数学形式化描述：

在词元级特征提取分支中，给定输入序列 $\mathbf{X} \in \mathbb{R}^{L \times d}$ ，其中 L 为输入序列长度， d 为 ERNIE 的嵌入维度，首先通过双向 LSTM 捕获上下文语义：

$$\overrightarrow{\mathbf{h}}_t = \text{LSTM}(\mathbf{x}_t, \overrightarrow{\mathbf{h}}_{t-1}) \quad (4.1)$$

$$\overleftarrow{\mathbf{h}}_t = \text{LSTM}(\mathbf{x}_t, \overleftarrow{\mathbf{h}}_{t-1}) \quad (4.2)$$

$$\mathbf{H} = [\overrightarrow{\mathbf{H}}; \overleftarrow{\mathbf{H}}] \in \mathbb{R}^{L \times 2h} \quad (4.3)$$

h 为单向 LSTM 隐藏单元数， $[;]$ 表示沿特征维度拼接。随后采用缩放点积注意力机制增强关键特征：

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d}} \right) \mathbf{V} \quad (4.4)$$

$\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{d \times d}$ 为可训练投影矩阵。层归一化模块通过仿射变换稳定训练：

$$\mathbf{x}' = \gamma \odot \frac{\mathbf{x} - \mu}{\sigma} + \beta \quad (4.5)$$

γ, β 为可学习缩放和平移参数， \odot 表示逐元素相乘。经过第二层 BiLSTM 进一步提取高阶特征后，通过全连接层降维：

$$\mathbf{Z} = \text{ReLU}(\mathbf{H}'\mathbf{W}_d + \mathbf{b}_d) \quad (4.6)$$

$\mathbf{W}_d \in \mathbb{R}^{2h \times 32}$ 为降维矩阵, \mathbf{H}' 为第二层 BiLSTM 输出。引入通道注意力模块动态调整特征重要性:

$$\mathbf{g} = \text{GAP}(\mathbf{Z}) = \frac{1}{L} \sum_{i=1}^L \mathbf{z}_i \quad (4.7)$$

$$\mathbf{a} = \sigma(\mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \mathbf{g})) \quad (4.8)$$

$$\mathbf{Z}' = \mathbf{Z} \odot \mathbf{a} \quad (4.9)$$

其中 GAP 表示 Global Average Pooling, $\mathbf{W}_1 \in \mathbb{R}^{32 \times 4}$ 为瓶颈层参数, σ 为 sigmoid 函数, \odot 表示通道维度的逐元素缩放。

句子级特征分支通过全局平均池化获取句子表征:

$$\bar{\mathbf{x}} = \frac{1}{L} \sum_{i=1}^L \mathbf{x}_i \in \mathbb{R}^d \quad (4.10)$$

\mathbf{x}_i 为 ERNIE 输出的第 i 个词元向量。逻辑回归层计算分类概率:

$$p_{\text{LR}} = \frac{1}{1 + \exp(-(\mathbf{w}^T \bar{\mathbf{x}} + b))} \quad (4.11)$$

$\mathbf{w} \in \mathbb{R}^d$ 为权重向量, b 为偏置项。

集成预测层融合双分支输出概率:

$$\hat{y}_{\text{final}} = 0.5 \cdot \sigma(\mathbf{w}_a^T \mathbf{Z}' + b_a) + 0.5 \cdot p_{\text{LR}} \quad (4.12)$$

$\mathbf{w}_a \in \mathbb{R}^{32}$ 为注意力分支的权重向量。最终通过阈值判定分类结果:

$$y_{\text{pred}} = \mathbb{I}(\hat{y}_{\text{final}} > 0.5) \quad (4.13)$$

$\mathbb{I}(\cdot)$ 为指示函数, 条件为真时取 1, 否则取 0。

该架构的创新点在于: (1) 通过双路结构同步捕捉局部序列模式与全局统计特征; (2) 引入通道注意力机制实现特征通道的自主校准; (3) 采用加权概率融合策略平衡模型复杂度与泛化性能。

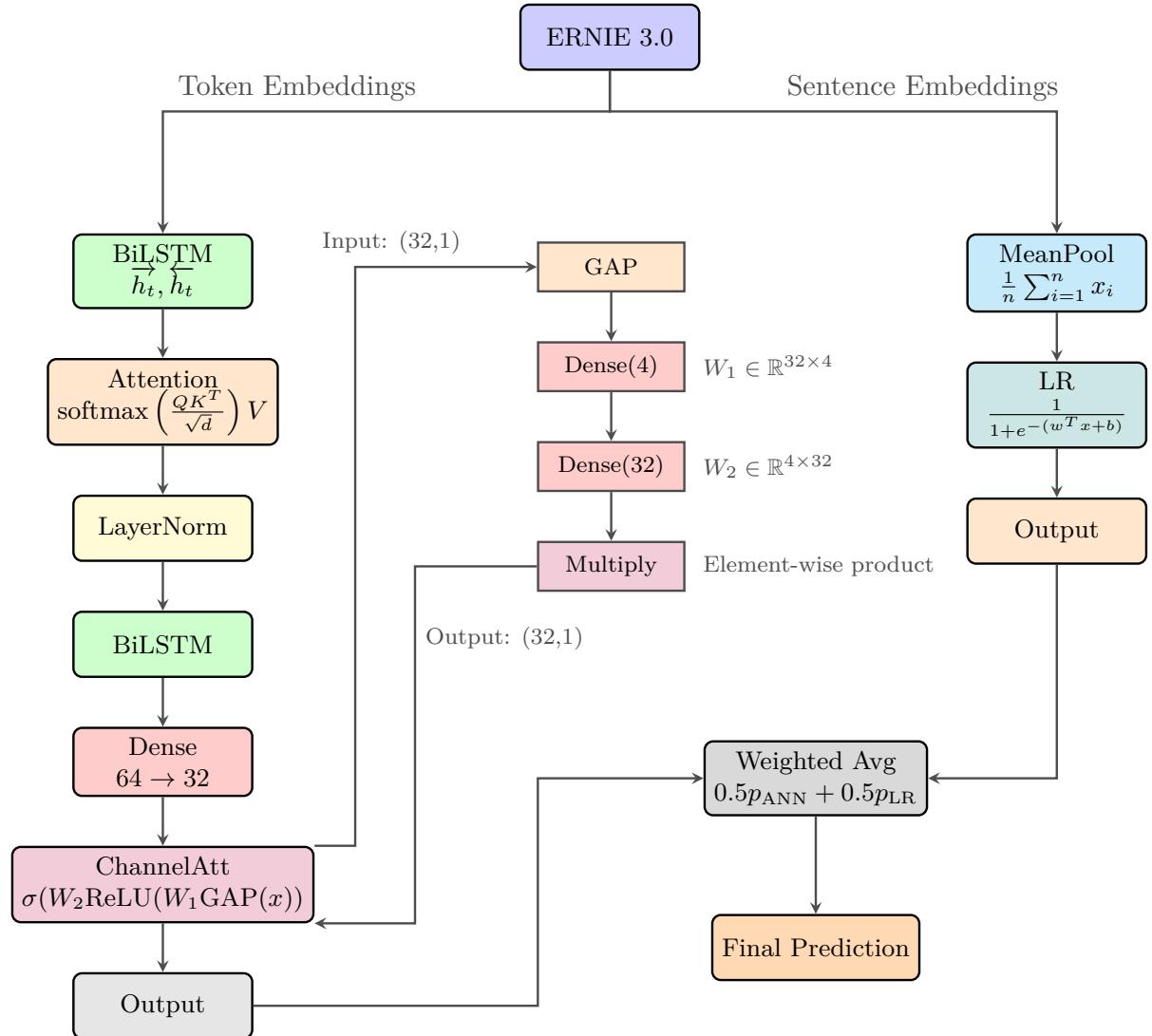


图 4.1 DepHybrid 架构组件图

接下来对所提出的 DepHybrid 架构进行了系统的算法复杂度分析，为评估其计算效率与可扩展性提供了理论依据。

DepHybrid 的时间复杂度主要集中于词元级特征分支。双向 LSTM 层的时间复杂度为 $O(Lh(d+h))$ ，其中 L 为序列长度， d 为嵌入维度， h 为隐藏单元数。自注意力机制的计算复杂度为 $O(L^2d + Ld^2)$ ，其主导项 L^2d 随序列长度呈二次增长，成为长序列处理的主要瓶颈。通道注意力模块通过全局平均池化与瓶颈结构设计，将复杂度控制在 $O(Ld + d^2/4)$ 。相较之下，句子级分支的复杂度较低，全局平均池化与逻辑回归分别仅需 $O(Ld)$ 和 $O(d)$ 时间。空间复杂度方面，双向 LSTM 占据主要部分 ($O(8h(d+h))$)，自注意力投影矩阵需 $O(3d^2)$ 空间。中间状态存储中，注意力矩阵的 $O(L^2)$ 空间需求在 $L > 512$ 时显著增加。通道注意力模块通过参数压缩，将存储需求降低至 $O(d^2/4)$ ，展现出良好的内存效率。

总体而言，该架构的时间复杂度为 $O(L^2d + Lh(d+h))$ ，空间复杂度为 $O(L^2 + 8h(d+$

h)。

4.2.2 UCB 参数优化算法

算法 1 基于 UCB 的逻辑回归参数优化算法

输入:

训练数据 $X_{\text{train}}, y_{\text{train}}$
 验证数据 $X_{\text{val}}, y_{\text{val}}$
 正则化系数候选集 $C_{\text{options}} = \{10^{-4}, 10^{-3}, \dots, 10^3\}$
 正则化类型候选集 $P_{\text{options}} = \{\ell_1, \ell_2\}$
 迭代次数 $T = 10$

输出: 最优参数组合 (C^*, p^*)

```

1: 初始化参数空间  $\mathcal{A} \leftarrow C_{\text{options}} \times P_{\text{options}}$ 
2: 创建统计数组 stats[N]  $\leftarrow \{\text{total\_reward} : 0.0, \text{count} : 0\}$ , 其中  $N = |\mathcal{A}|$ 
3: for  $t = 1$  to  $T$  do
4:   if 存在未探索参数 then
5:     选择第一个满足 stats[i].count = 0 的参数  $a_i$ 
6:   else
7:     for  $i = 1$  to  $N$  do
8:       计算平均奖励  $\hat{\mu}_i \leftarrow \text{stats}[i].\text{total\_reward}/\text{stats}[i].\text{count}$ 
9:       计算 UCB 值  $\text{UCB}_i \leftarrow \hat{\mu}_i + \sqrt{2 \ln t / \text{stats}[i].\text{count}}$ 
10:    end for
11:    选择  $a^* \leftarrow \operatorname{argmax}_{a_i} \text{UCB}_i$ 
12:  end if
13:  if 参数组合有效 then
14:    训练模型  $\mathcal{M} \leftarrow \text{LogisticRegression}(C = a^*.C, \text{penalty} = a^*.p)$ 
15:     $\mathcal{M}.\text{fit}(X_{\text{train}}, y_{\text{train}})$ 
16:     $\hat{y} \leftarrow \mathcal{M}.\text{predict}(X_{\text{val}})$ 
17:    计算奖励  $r \leftarrow \text{F1-Score}(y_{\text{val}}, \hat{y})$ 
18:  else
19:    设置惩罚奖励  $r \leftarrow -1$ 
20:  end if
21:  更新统计量:
22:  stats[a*].total_reward  $\leftarrow \text{stats}[a^*].\text{total\_reward} + r$ 
23:  stats[a*].count  $\leftarrow \text{stats}[a^*].\text{count} + 1$ 
24: end for
25: 计算最终平均奖励  $\mu_i^* \leftarrow \text{stats}[i].\text{total\_reward}/\text{stats}[i].\text{count} \quad \forall i$ 
26: 选择最优参数  $i^* \leftarrow \operatorname{argmax}_i \mu_i^*$ 
27: return  $a_{i^*}$ 

```

针对 DepHybrid 架构的超参数优化问题，本文提出一种基于上置信区间（UCB）的强化学习调优算法。该算法将参数选择建模为多臂老虎机问题，通过平衡探索（exploration）与利用（exploitation）实现高效参数搜索。本文以逻辑回归为例进行参数调优，定义参数空间为：

$$\mathcal{A} = \{(C, p) \mid C \in \{10^{-4}, 10^{-3}, \dots, 10^3\}, p \in \{\ell_1, \ell_2\}\} \quad (4.14)$$

其中 C 为正则化系数， p 为惩罚项类型，共构成 $8 \times 2 = 16$ 种参数组合。每个参数组合视为一个“臂”，其期望奖励通过验证集 F1 分数评估。

UCB 选择策略的数学表达为：

$$a_t = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left[\hat{\mu}_a + \sqrt{\frac{2 \ln t}{n_a}} \right] \quad (4.15)$$

其中 $\hat{\mu}_a$ 为参数组合 a 的历史平均奖励， n_a 为选择次数， t 为当前迭代次数。设参数空间大小为 N ，迭代次数为 T ，则时间复杂度为 $O(TN + T \cdot t_{\text{train}})$ ，其中 t_{train} 为单次训练耗时，空间复杂度为 $O(N)$ ，仅需存储各参数的统计量。基于 UCB 的逻辑回归参数优化流程如算法1。

4.3 实验目的

本研究在构建抑郁识别模型时需重点考量以下核心要素：特征工程作为模型学习的输入基础需确保文本特征的显著性，模型架构需优化特征到决策的映射效率，任务设计需合理约束模型的优化目标。具体而言，实验设计需解决三个层面的关键问题：在特征层面需验证文本特征的判别有效性，在模型层面需探索高性能的结构设计，在数据层面需筛选适配的访谈问题数据集。

此外，实验实施过程中需着重处理若干技术细节：首先需建立科学的基准对照体系，从多变量实验中确定有效的基线模型；其次为避免特征空间、模型架构与任务目标间的隐性耦合效应对实验结果的影响，需通过逐步排除法剔除性能不足的自变量；最后需构建系统化的超参数优化策略，通过自动化调参方法提升模型性能。这三个维度的协同优化构成本文提升抑郁识别精度的核心研究路径。

而精神疾病的诊断涉及复杂的临床决策链，模型需提供可追溯的特征权重分布与决策逻辑，使精神科医生能够验证文本特征（如情感词汇频率、句法结构异常）与抑郁评估量表之间的病理关联。因此医疗场景的特殊性要求必须将可解释人工智能（XAI）也纳入核心考量。

4.4 实验设计

本研究通过控制变量法全面探究特征、模型、数据层面对抑郁识别性能的影响。

首先，特征表示层面采用 BoW、BERT、BERT 微调、ERNIE、RoBERTa 及 eHealth 模型。BoW 作为基线方法可捕捉显性词汇统计特征；BERT 及其变体通过 Transformer 架构提取深度上下文语义表征，微调策略实现预训练语言模型与抑郁语义空间的精准对齐；ERNIE 增强医疗实体识别能力，RoBERTa 优化长文本建模效果；eHealth 领域自适应模型则通过医疗健康语料预训练获得的领域特异性表示可有效捕捉抑郁表述中的病理学特征。

其次，分类模型架构层面采用机器学习与深度学习两种类型的方法，其中基线机器学习方法 (LR、SVM、k-NN、NBC、DT、GP、PAA、RF、Extra tree、AdaBoost、GBDT、XGBoost、LightGBM、HGB、CatBoost、MLP) 覆盖监督学习主要范式，构建了一个全面的基线系统，确保实验结果的可信度，为后续深度学习模型提供对比基础。深度学习则通过设计 FNN 结构 (SFC、FC-Drop、Conv-Pool、CAM、Flatten、ResLn)、RNN 结构 (LSTM、BiLSTM、GRU、BiGRU) 与 Seq2Seq 结构 (SAM、MSA、LayerNorm) 的模块进行相互组合，探究各组合模块间的性能。通过跨架构组合验证抑郁表征的普适性假设，量化评估各组件对模型偏差-方差均衡的贡献度，这种组合策略可验证 FNN 对静态特征的提取效能、检验 RNN 对时序依赖的建模能力、探索 Transformer 架构在心理状态建模中的迁移适应性。

在研究任务设计层面，本文采用分项实验设计策略，基于结构化访谈中的独立问题构建抑郁症自动检测模型，通过对比性实验系统评估不同访谈问题在抑郁症判别中的特征表征能力与分类贡献度，从而量化解析各语义单元对心理健康评估的预测效力。

在参数优化中选择 UCB 算法对参数进行调优，该方法适用于需要平衡尝试未知选项和利用已知最优选项的情况，在有限的计算资源下提供了高效的优化路径。在模型可迁移性与领域适应性研究层面，本研究构建跨问题双向推理架构，采用自监督学习机制实现访谈问题间的应答内容互预测（如通过问题 A 的语义表征推断问题 B 的应答模式），系统揭示抑郁评估中多轮对话的潜在语义拓扑结构。在模型可解释性层面，t-SNE 从宏观特征空间结构验证模型表征学习能力，SHAP 则从微观特征贡献度解析模型决策合理性。因此通过 t-SNE 和 SHAP 方法可视化特征分布及特征重要程度，验证模型架构的可信度。

在性能评估中，本文使用分层十折交叉验证，通过平均值和标准差联合表示模型性能指标 (Mean \pm SD)。分层抽样保持每折数据的类别分布与总体一致，这对抑郁识别中普遍存在的样本不平衡问题具有关键纠偏作用；平均值反映模型的平均性能，而标准差显示性能的波动情况，两者结合可以全面评估模型的稳定性和泛化能力。

本研究中的实验对比分析需特别注意：关注重点应放在分类性能指标的相对差异和排序关系上，而非绝对数值本身。这主要是因为最终的分类准确度指标会受到多重变量的影响，包括但不限于算法模型的选取差异、超参数配置的调整幅度，以及十折交叉验证过程中因数据划分随机性产生的波动效应。这种实验设计层面的变量特性决定了不同实验批次间的绝对数值不具备直接可比性，而相对排序关系和指标间的比例特征则能更稳定地反映模型的真实性能差异。

4.4.1 文本特征类型设计

1. 词袋模型特征

本文词袋模型使用的特征如表4.1所示。在抑郁情感分析任务中，句子长度、标点符号、词性统计、停用词比例及 TF-IDF 特征的选取，源于对抑郁语言模式的深层解析与临床心理学理论的结合。抑郁倾向文本常通过句子长度的两极分化（短句碎片化或冗长重复）反映认知阻滞或情绪泛滥，而标点符号的非常规使用（如高频省略号暗示表达中断、感叹号密集映射情绪失控）则成为情感外化的直接标记。词性分布上，第一人称代词过度集中揭示自我沉溺倾向，负面形容词聚类与动词时态单一化分别指向情感维度窄化与创伤反刍，名词/动词的低多样性进一步印证认知灵活性的退化。停用词比例的异常波动（功能词冗余或匮乏）从心理语言学角度映射语言组织能力衰退或情感麻木状态，而 TF-IDF 加权的抑郁特异性词汇（如显性负面词、生理隐喻及认知扭曲标记）通过逆文档频率强化临床相关性，有效区分通用负面情绪与病理性表达。

表 4.1 词袋模型提取特征详细描述

特征名称	维数	特征说明
length	1	句子长度, 统计每一条访谈文本中字符的个数
punctuation_features	22	标点符号数, 统计每一条访谈文本中。!?,;‘’“”()[]{}<>《》* 的数量
pos_features	3	词性统计, 统计每一条访谈文本中名词、动词及形容词的个数
stopword_frequency	1	停用词比例, 统计每一条访谈文本中的、了、是、在、和、有、我、他、她、它、这、那的比例
X_tfidf	1864	文档 TF-IDF 值, 统计每一个词在整个访谈文本中的 TF-IDF 值

2. 预训练模型语言模型特征嵌入

表 4.2 预训练语言模型类型详细描述

模型类别	模型名称	模型说明
BERT	bert-base-chinese ^[84]	BERT-Base-Chinese 是基于 BERT 架构的中文预训练模型，采用双向 Transformer 编码器结构，包含 12 个隐藏层、768 维隐藏状态及 12 个自注意力头（参数量 1.1 亿）。通过中文维基百科数据，结合掩码语言模型和下一句预测任务进行预训练，适用于多种中文语境。
ERNIE	ernie-3.0-base-zh ^[108]	ERNIE-3.0-Base-Zh 是百度研发的知识增强型中文预训练模型，基于 Transformer 架构构建（12 层/768 维隐藏层/12 个注意力头），参数量 1.1 亿。模型通过数千亿词级的百科、书籍及互联网多源语料训练，创新性地融合传统掩码预测任务与实体关系推理、语义分类等知识增强任务，并支持词/短语/句子多粒度语义建模。
RoBERTa	chinese-roberta-wwm-ext ^[109]	Chinese-RoBERTa-wwm-ext 是由哈工大团队研发的中文预训练模型。基于 RoBERTa 架构，采用 12 层 Transformer 编码器（768 隐藏层/12 头，1.1 亿参数）。核心创新是通过全词掩码策略，将中文词汇作为整体单元遮蔽，强化对词汇边界和语义完整性的理解。模型使用海量多领域中文语料训练，并应用动态掩码等优化技术提升性能。
eHealth	ernie-health-zh ^[110]	ERNIE-Health-Zh 是百度开发的医疗领域中文模型，基于 ERNIE 框架深度优化，专注处理医学文献、电子病历等专业文本。通过医疗领域大规模预训练，显著提升中文医疗 NLP 任务的语义理解和处理能力。

本文使用的预训练语言模型类型如表4.2所示。在预训练模型嵌入处理中，LSTM 等循环网络通过时序门控机制逐词元传递隐藏状态，使每个 token 的嵌入（如双向 LSTM 的前后向状态拼接）能够动态捕捉上下文依赖关系——这种细粒度建模对抑郁文本中的矛盾语义流变（如“想消失却又害怕死亡”中的转折逻辑）、情绪渐进累积（连续短句的压抑感叠加）等时序模式至关重要。而全连接网络等静态架构因固定维度输入限制，需通过全局平均池化将变长序列压缩为句嵌入，本质是以词序信息丢失为代价实现维度统一（如忽略“绝望中挣扎”与“挣扎中绝望”的语义差异），转而依赖词袋层面的统计特征（如负面词密度）进行粗粒度分类。这种设计差异映射了模型能力与任务需求的匹配：序列模型通过 token 级嵌入解析抑郁语言中的微观动态（如自我否定的渐进强化），虽计算成本较高却适

配深度语义分析；非序列模型通过句嵌入的高效性支持大规模筛查，但难以捕捉临床诊断关注的时序行为标记（如情绪波动轨迹）。二者在应用中常形成互补——LSTM 提取局部异常模式（如特定标点与代词的组合），句嵌入提供全局情感基调，共同构建多层次抑郁语言表征体系。

4.4.2 神经网络结构设计

为了更好地完成本研究的抑郁识别任务，本文使用了以下神经网络模块，计划通过相应的组合，以实现更好的性能。

1. 简单全连接模块

简单全连接（SFC）模块由输入层、含 32 个 ReLU 神经元的隐藏层（Hidden）和输出层（Dense）构成（图4.2）。输入层接收状态向量 $\mathbf{x} \in \mathbb{R}^s$ ，隐藏层执行非线性变换：

$$\mathbf{h} = \text{ReLU}(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) \quad (4.16)$$

其中 $\mathbf{W}_1 \in \mathbb{R}^{32 \times s}$, $\mathbf{b}_1 \in \mathbb{R}^{32}$ 。输出层通过 Sigmoid 函数实现二分类：

$$\hat{y} = \sigma(\mathbf{W}_2 \mathbf{h} + \mathbf{b}_2) \quad (4.17)$$

参数为 $\mathbf{W}_2 \in \mathbb{R}^{1 \times 32}$, $\mathbf{b}_2 \in \mathbb{R}$ 。

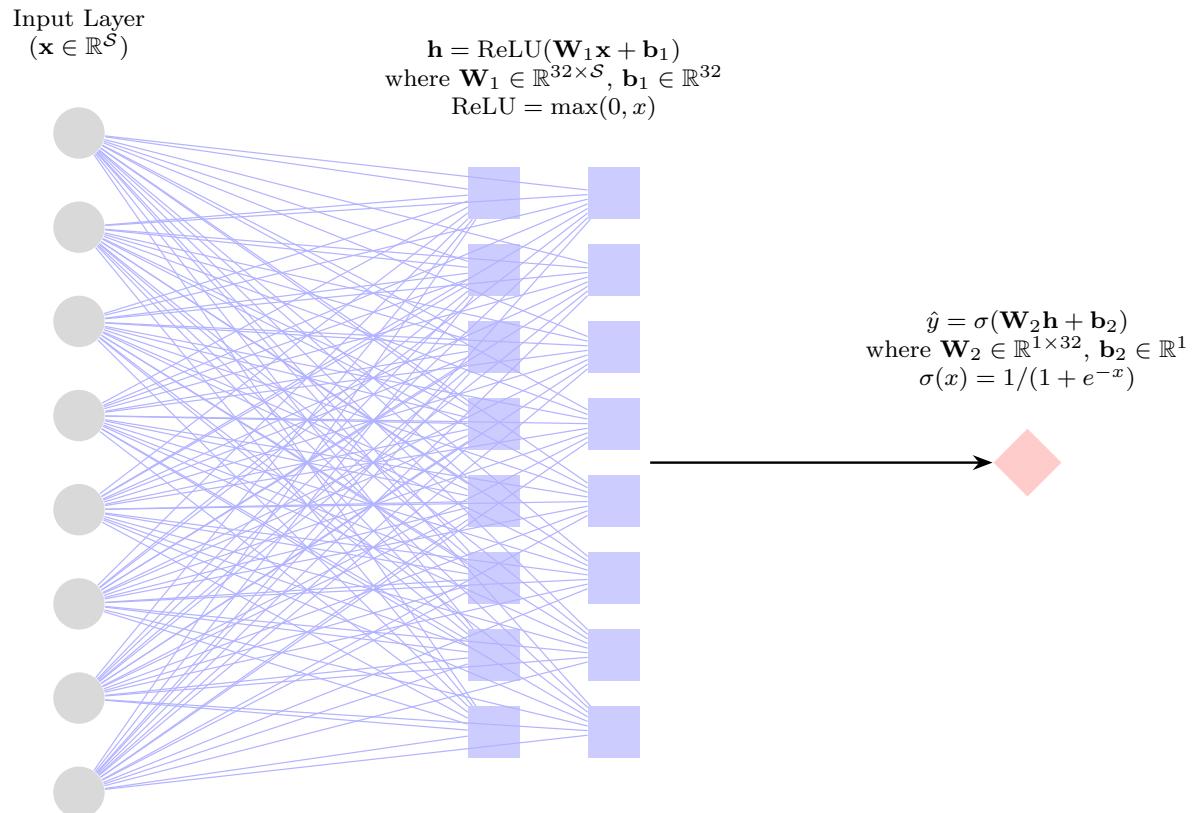


图 4.2 SFC 模块，其中 s 为特征维数

2. 全连接-暂退模块

FC-Drop 模块（图4.3）由输入层、全连接隐藏层和 Dropout 层构成（移除 Dropout 则为 FC 模块）。输入特征 $\mathbf{x} \in \mathbb{R}^S$ 通过权重矩阵 $\mathbf{W}_1 \in \mathbb{R}^{64 \times S}$ 映射至 64 维隐层，经 ReLU 激活生成特征表示 $\mathbf{h} = \text{ReLU}(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$ 。训练时 Dropout 层以概率 $\theta \in \{0.2, 0.3\}$ 对隐层施加随机掩码 $\epsilon \sim \text{Bernoulli}(1 - \theta)$ ，输出为 $\hat{y} = \sigma(\mathbf{W}_2(\mathbf{h} \odot \epsilon) + \mathbf{b}_2)$ ，该机制通过阻断神经元协同适应增强泛化能力。

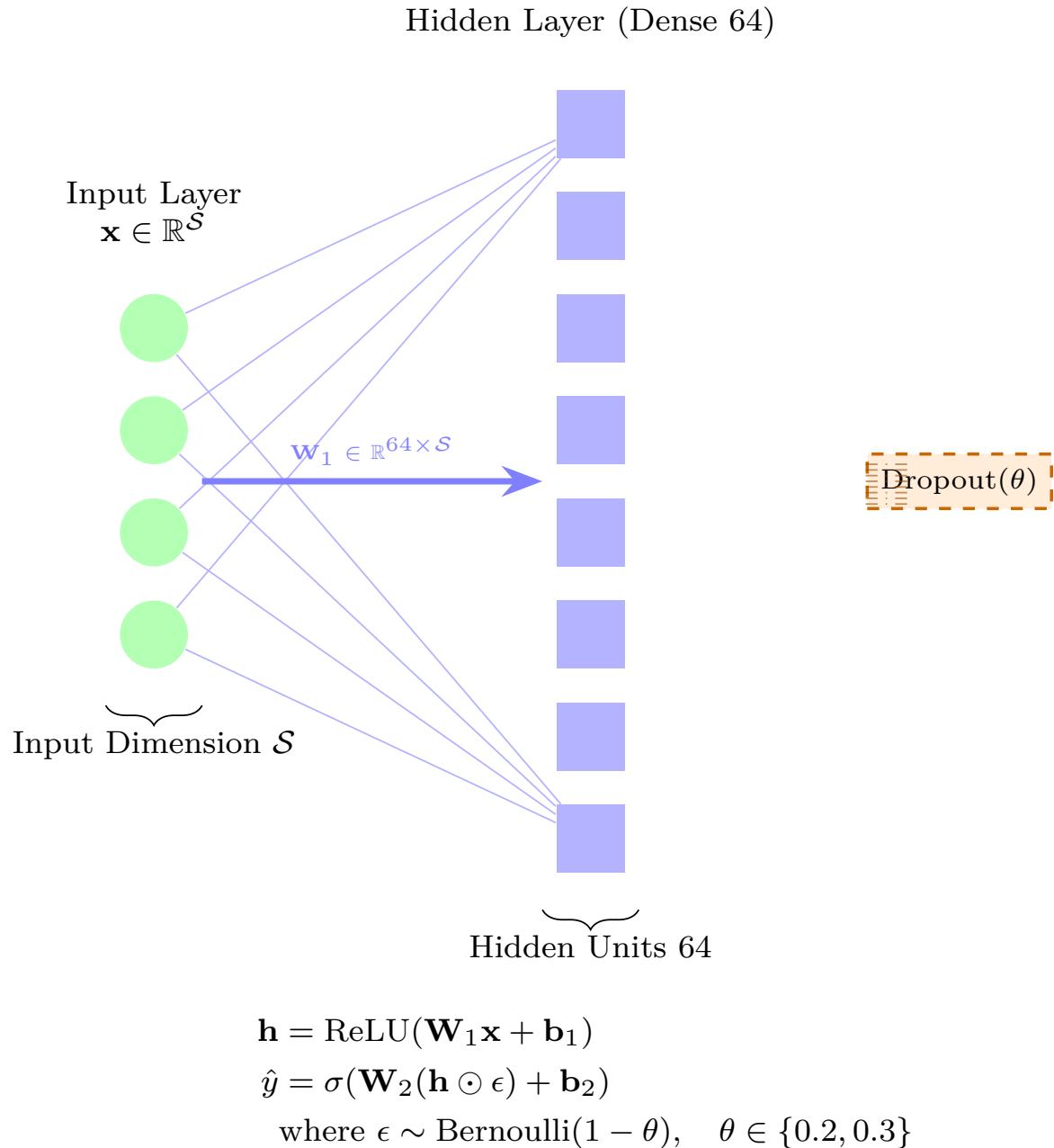


图 4.3 FC-Drop 模块，其中 S 为特征维数， \odot 表示元素乘， θ 为丢弃隐藏单元概率， σ 为 sigmoid 函数

3. 卷积-池化模块

卷积-池化 (Conv-Pool) 模块由输入层、Conv1D 层和 MaxPooling1D 层组成 (图4.4)。输入数据维度为 $L \times C_{\text{in}}$ (时间步长 \times 输入通道)。卷积层使用 n 个三维滤波器 $\mathbf{W} \in \mathbb{R}^{3 \times C_{\text{in}} \times n}$ 进行特征提取，通过带 ReLU 激活的滑动卷积：

$$y_j^{(i)} = \text{ReLU}\left(\sum_{k=0}^2 W_k \cdot x_{j+k}^{(i)} + b\right) \quad (4.18)$$

采用 same padding 保持输出长度 L ，得到 $L \times n$ 特征映射。最大池化层通过步长 2 的下采样：

$$z_j^{(i)} = \max_{k \in \{0,1\}} (y_{2j+k}^{(i)}) \quad (4.19)$$

输出维度压缩为 $\lfloor L/2 \rfloor \times n$ ，实现特征降维。滤波器数量 n 可配置为 64 或 128。

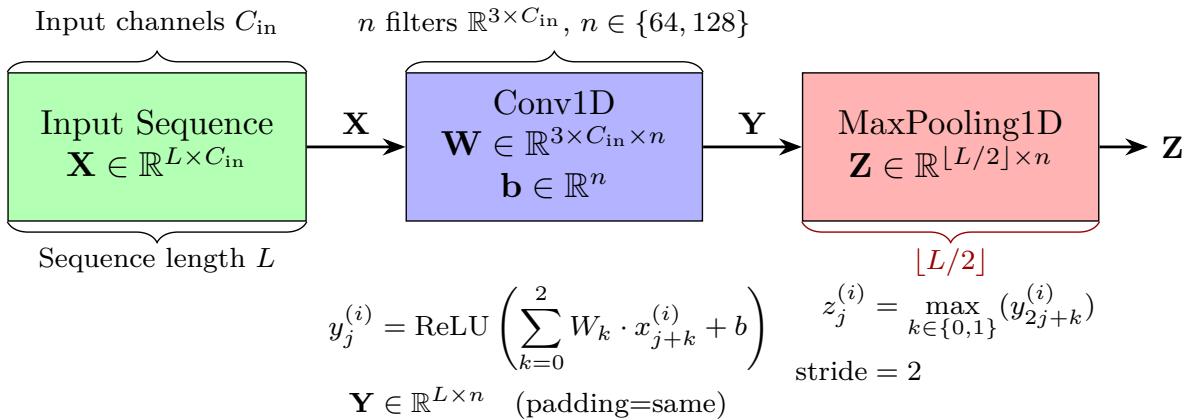


图 4.4 Conv-Pool 模块，其中 C_{in} 为特征通道数， n 为滤波器个数， L 表示时间序列长度

4. 通道注意力机制模块

通道注意力机制 (CAM) 模块 (图4.5) 基于 SE 模块^[111] 设计，通过三阶段流程对输入特征 $\mathbf{X} \in \mathbb{R}^{L \times F}$ 进行动态通道权重学习。首先，特征压缩阶段通过全局平均池化提取通道全局信息，生成统计量 $\mathbf{s} = \frac{1}{L} \sum_{i=1}^L \mathbf{X}^{(i)} \in \mathbb{R}^F$ ；随后在非线性变换阶段，采用降维因子为 8 的两层全连接网络 ($\mathbf{W}_1 \in \mathbb{R}^{(F/8) \times F}$ 与 $\mathbf{W}_2 \in \mathbb{R}^{F \times (F/8)}$) 进行特征交互，通过 $\mathbf{z} = \text{ReLU}(\mathbf{W}_1 \mathbf{s} + \mathbf{b}_1)$ 降维后，经 $\mathbf{A} = \sigma(\mathbf{W}_2 \mathbf{z} + \mathbf{b}_2)$ 升维生成通道注意力权重；最终在特征重标定阶段，通过 $\hat{\mathbf{X}}^{(i)} = \mathbf{X}^{(i)} \odot \mathbf{A}$ 的逐通道乘积实现特征增强，其中 Sigmoid 函数 $\sigma(\cdot)$ 确保注意力权重 $\mathbf{A} \in [0, 1]^F$ 的归一化。该模块通过分离主路径与注意力路径，实现对关键通道特征的自适应强化。

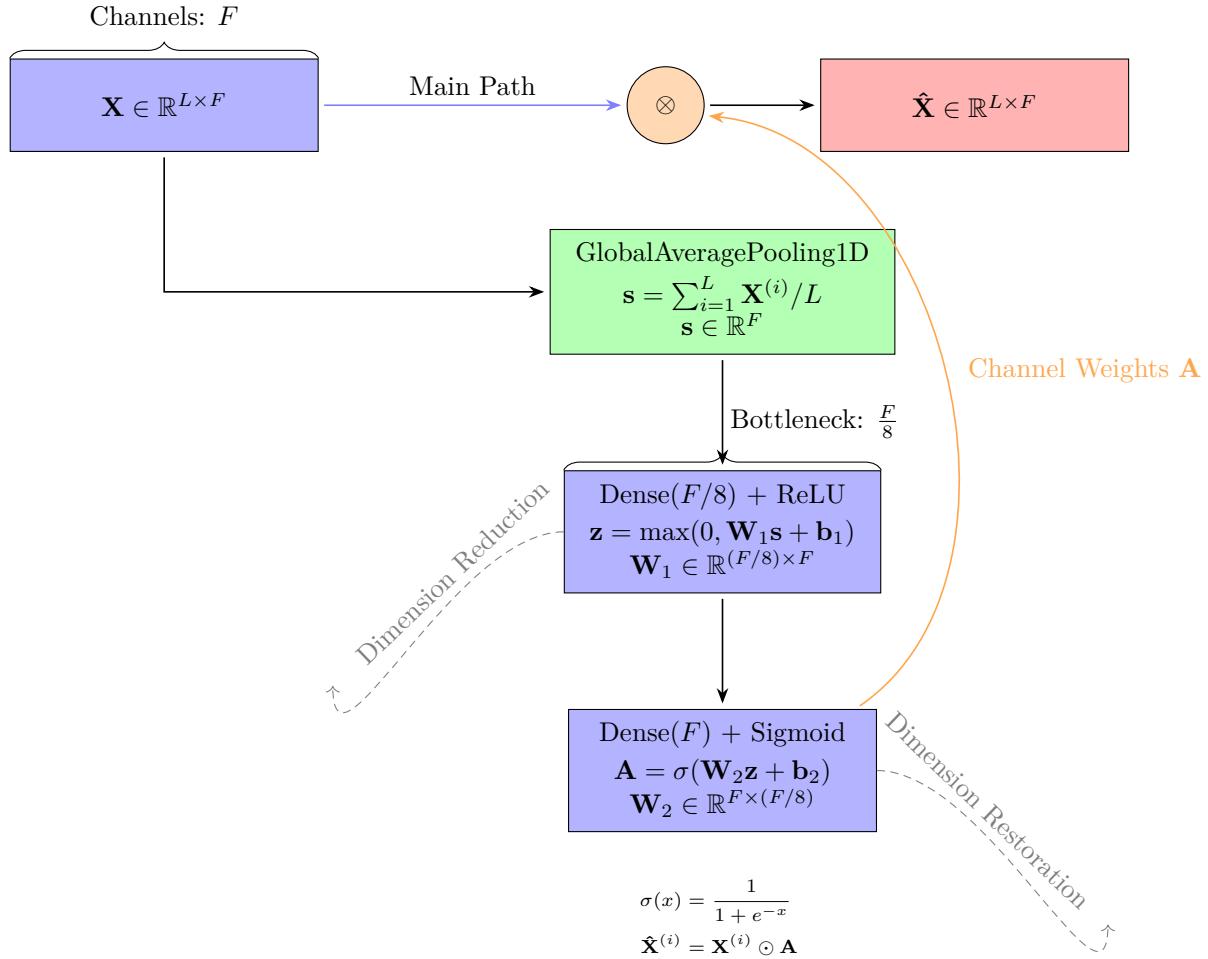


图 4.5 CAM 模块，其中 F 为特征通道数， \odot 表示元素乘， L 表示序列长度

5. 展平模块

展平 (Flatten) 模块将多维输入转换为一维特征向量，便于全连接层处理。对于输入张量 $\mathbf{X} \in \mathbb{R}^{H \times W \times C}$ ，经过 32 个 3×3 卷积（步长 $s=1$ ）后输出 $\mathbf{X}' \in \mathbb{R}^{H' \times W' \times 32}$ ，其中 $H' = (H-3)//s+1$ (W' 同理)。展平操作将其转换为一维向量：

$$\text{Flatten}(\mathbf{X}') = [x_{111}, \dots, x_{H'W'32}]^\top \in \mathbb{R}^D \quad (4.20)$$

其中 $D = H' \times W' \times 32$ ，该操作在保留空间特征关联性的同时实现三维到一维的维度转换，为后续全连接层提供结构化输入。

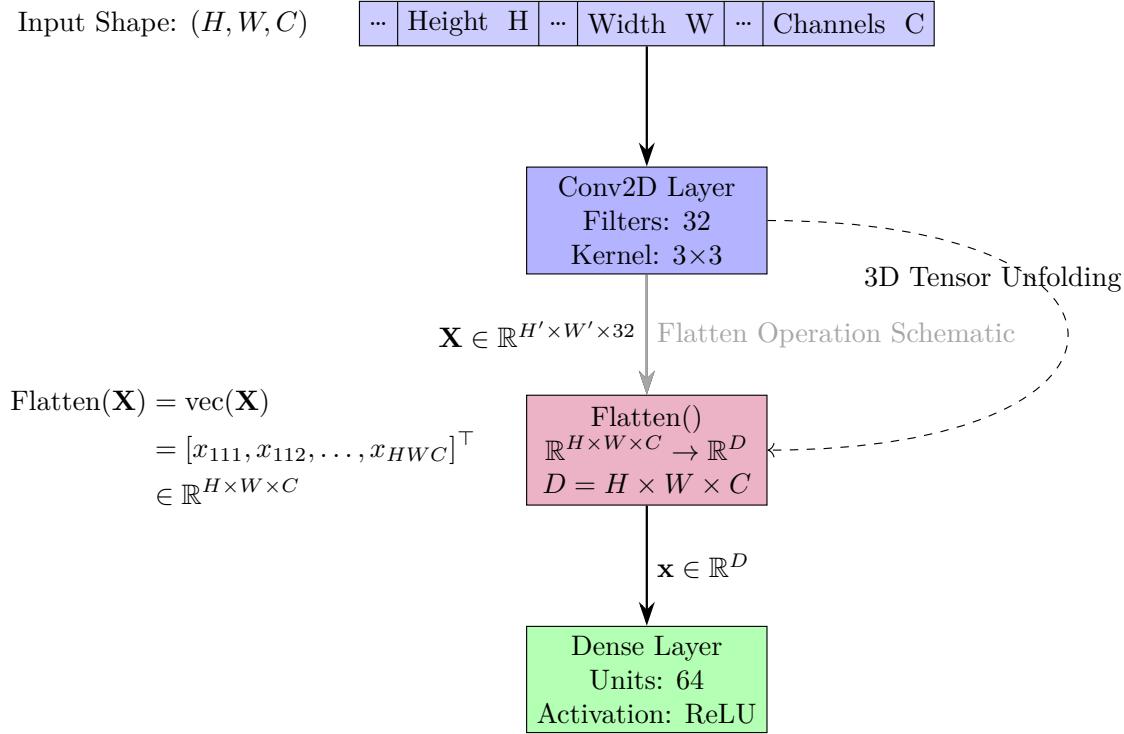


图 4.6 Flatten 模块示意图

6. 残差-池化模块

残差学习模块 (ResLn)^[112] (图4.7) 采用双路径结构实现稳定梯度传播。输入特征 $\mathbf{X} \in \mathbb{R}^{L \times C_{\text{in}}}$ 通过主路径进行两级卷积处理：第一层 $\mathbf{W}_1 \in \mathbb{R}^{3 \times C_{\text{in}} \times F}$ 扩展通道维度，第二层 $\mathbf{W}_2 \in \mathbb{R}^{3 \times F \times F}$ 保持特征维度；跳跃路径通过条件卷积实现通道匹配：

$$\mathcal{G}(\mathbf{X}) = \begin{cases} \mathbf{X} & C_{\text{in}} = F \\ \mathbf{W}_p * \mathbf{X} & C_{\text{in}} \neq F \end{cases} \quad (\mathbf{W}_p \in \mathbb{R}^{1 \times C_{\text{in}} \times F}) \quad (4.21)$$

特征融合采用残差加法：

$$\mathbf{X}' = \text{ReLU}(\mathcal{F}(\mathbf{X}) + \mathcal{G}(\mathbf{X})) \quad (4.22)$$

其中主路径 $\mathcal{F}(\mathbf{X}) = \mathbf{W}_2 * (\text{ReLU}(\mathbf{W}_1 * \mathbf{X}))$ ，延迟激活操作保留残差信息完整性。跳跃连接建立跨层梯度通路，缓解浅层参数 \mathbf{W}_1 的梯度衰减。后续连接最大池化层构成 ResLn-Pool 模块。

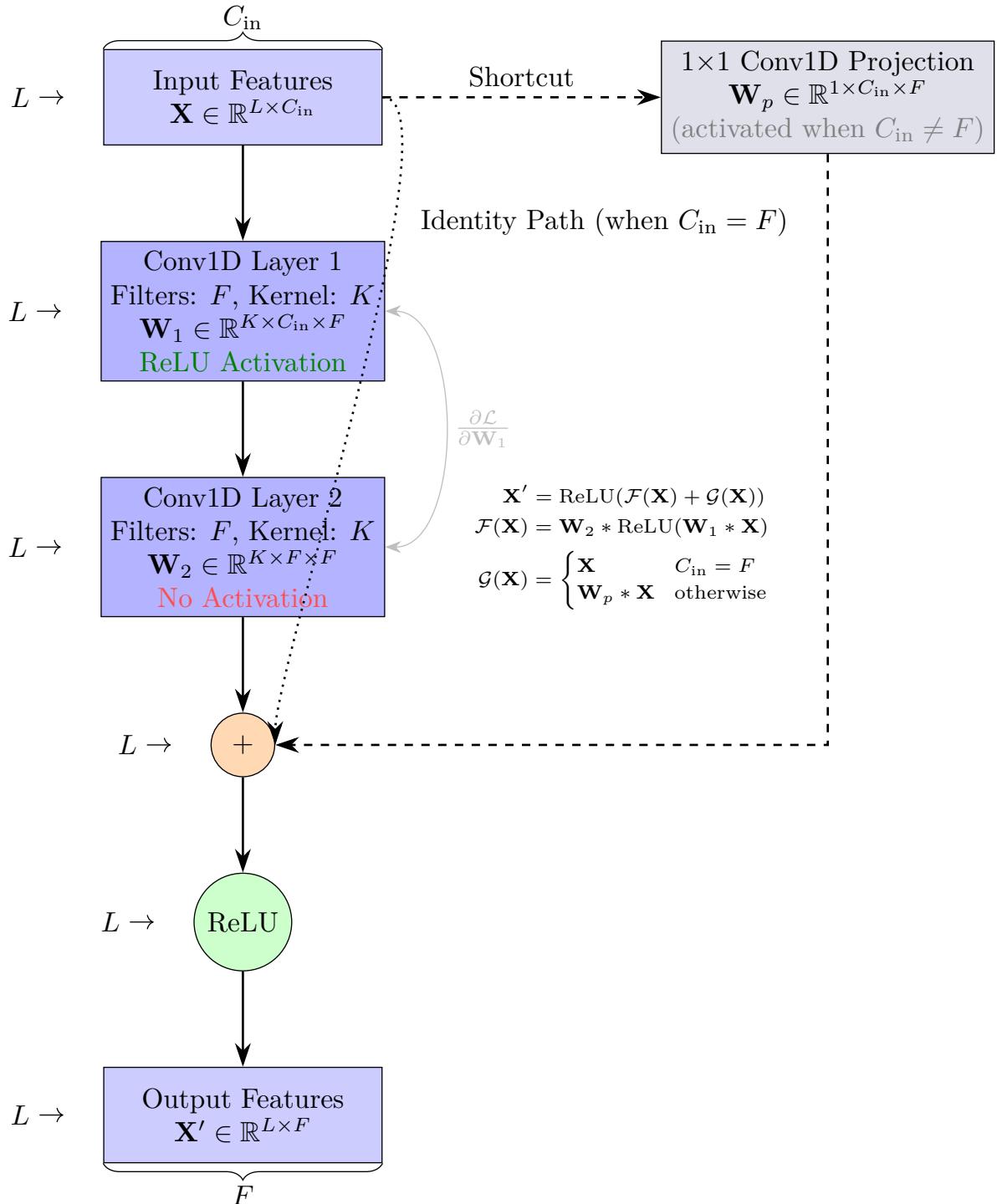
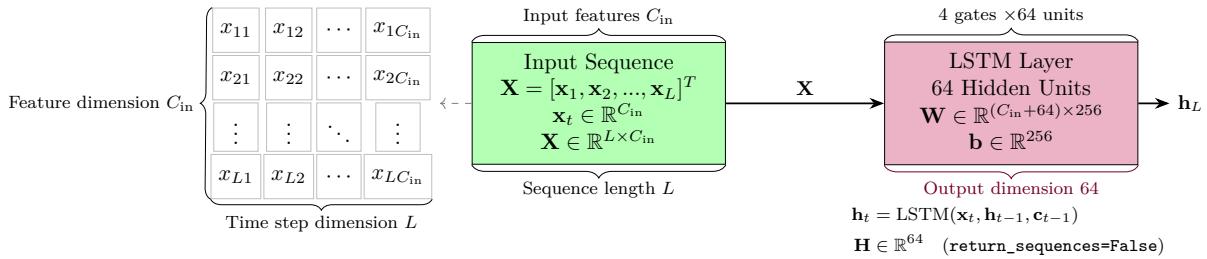


图 4.7 ResLn 模块，其中 C_{in} 为特征通道数， F 表示滤波器个数， \mathcal{L} 为损失函数， L 表示序列长度， $K = 3$ 为卷积核大小

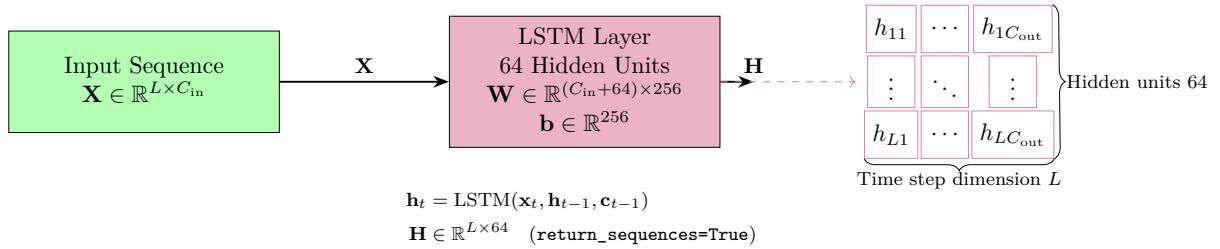
7. 长短期记忆模块

长短期记忆 (LSTM) 模块 (图4.8) 输入为时序特征矩阵 $\mathbf{X} \in \mathbb{R}^{L \times C_{\text{in}}}$ (L 为时间步长)。每个时间步 \mathbf{x}_t 经全连接层输入至含 64 个隐藏单元的 LSTM 层，其参数矩阵 $\mathbf{W} \in \mathbb{R}^{(C_{\text{in}}+64) \times 256}$

包含输入门、遗忘门、输出门和候选记忆单元四组参数（每组 64 个单元）。序列末端输出模式（`return_sequences=False`）返回末状态 $\mathbf{h}_L \in \mathbb{R}^{64}$ ；全输出模式（`return_sequences=True`）生成状态矩阵 $\mathbf{H} \in \mathbb{R}^{L \times 64}$ 。门控机制通过 $\mathbf{h}_t = \text{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1}, \mathbf{c}_{t-1})$ 实现记忆更新，其中 \mathbf{c}_{t-1} 为前一细胞状态。



(a) LSTM(A) 模块, 序列末端输出模式

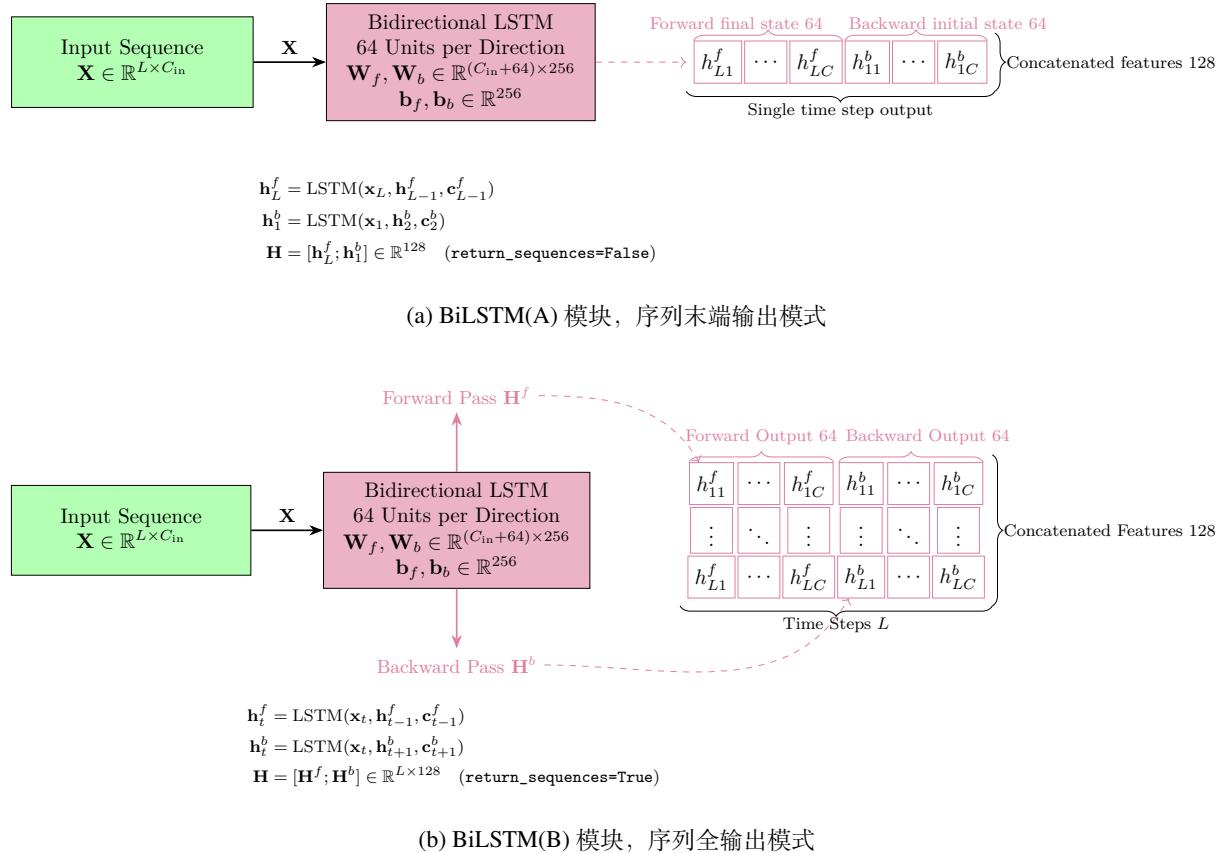


(b) LSTM(B) 模块, 序列全输出模式

图 4.8 LSTM 模块, 其中 C_{in} 为特征维度, L 表示时间步长维度

8. 双向长短期记忆模块

双向长短期记忆（BiLSTM）模块以 $L \times C_{\text{in}}$ 维时序数据 \mathbf{X} 作为输入，通过独立配置参数的双向 LSTM 通道（前向/反向各含 64 个隐藏单元）捕获上下文特征。其输出模式分为两类：非序列输出模式下，网络提取前向通道末端状态 $\mathbf{h}_L^f \in \mathbb{R}^{64}$ 与反向通道起始状态 $\mathbf{h}_1^b \in \mathbb{R}^{64}$ ，拼接形成 128 维全局语义向量 $\mathbf{H} = [\mathbf{h}_L^f; \mathbf{h}_1^b]$ ，前/后半部分分别表征历史累积与未来起始的上下文信息；序列输出模式下，则生成时间步对齐的前向状态矩阵 $\mathbf{H}^f \in \mathbb{R}^{L \times 64}$ 和反向状态矩阵 $\mathbf{H}^b \in \mathbb{R}^{L \times 64}$ ，通过列拼接构建 $L \times 128$ 维增强特征矩阵 $\mathbf{H} = [\mathbf{H}^f \parallel \mathbf{H}^b]$ ，使每个时间步同时融合前向历史与反向未来信息。该设计通过双向特征融合强化时序建模能力，非序列模式聚焦整体语义提取，序列模式保留局部上下文关联性。

图 4.9 BiLSTM 模块，其中 C_{in} 为特征维度， L 表示时间步长维度

9. 门控循环模块

门控循环单元 (GRU) 通过动态门控机制实现高效时序建模 (图4.10)，其输入为 $L \times C_{\text{in}}$ 维时序数据 \mathbf{X} ，内部参数矩阵 $\mathbf{W} \in \mathbb{R}^{(C_{\text{in}}+64) \times 192}$ (包含更新门 \mathbf{W}_z 、重置门 \mathbf{W}_r 和候选状态 \mathbf{W}_h 各 64 个单元) 及偏置 $\mathbf{b} \in \mathbb{R}^{192}$ ，通过门控计算逐时间步更新隐藏状态：更新门 $\mathbf{z}_t = \sigma(\mathbf{W}_z[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z)$ 控制历史信息保留比例，重置门 $\mathbf{r}_t = \sigma(\mathbf{W}_r[\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r)$ 调节候选状态 $\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h[\mathbf{r}_t \odot \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_h)$ 的生成，最终状态 $\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \tilde{\mathbf{h}}_t$ 融合新旧信息；输出支持单步模式 (末状态 $\mathbf{h}_L \in \mathbb{R}^{64}$) 或序列模式 (全时序状态矩阵 $\mathbf{H} \in \mathbb{R}^{L \times 64}$ ，每列 \mathbf{h}_t 编码 t 时刻的上下文特征)。

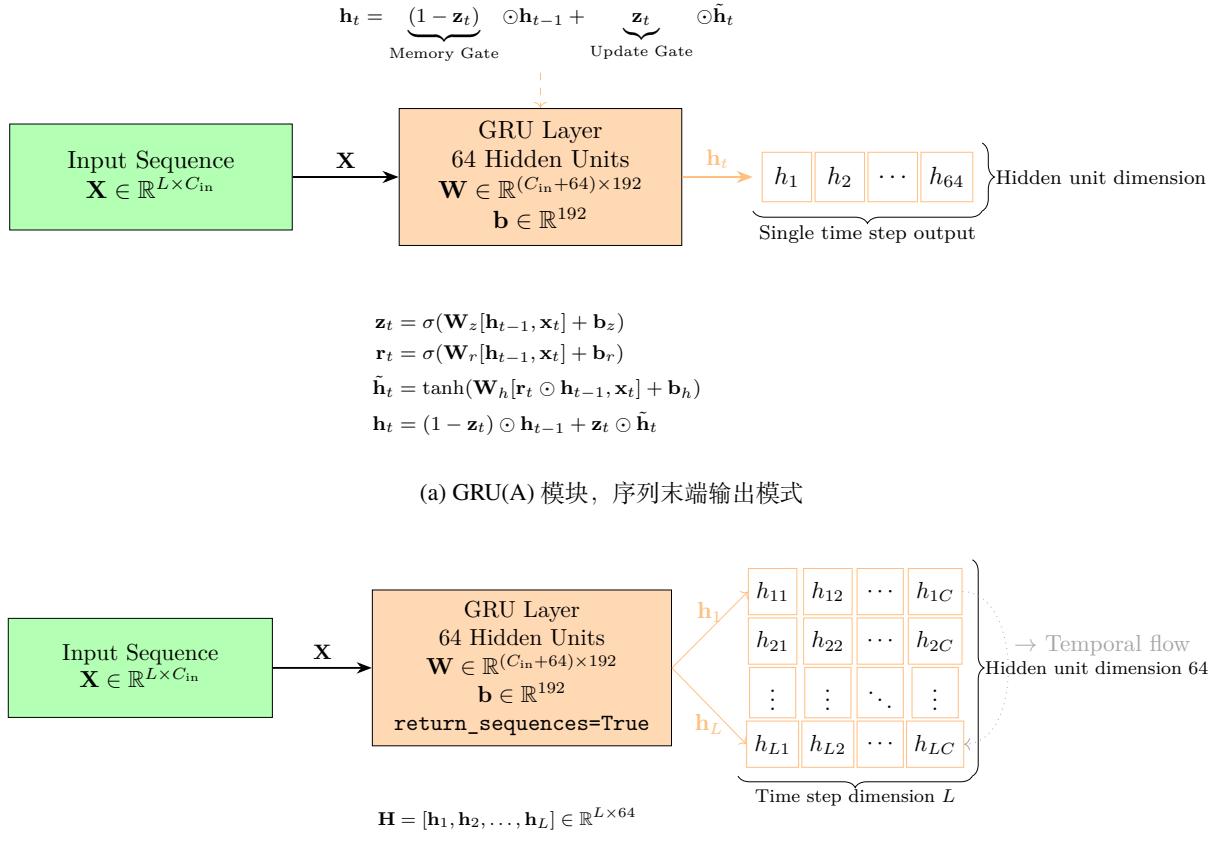
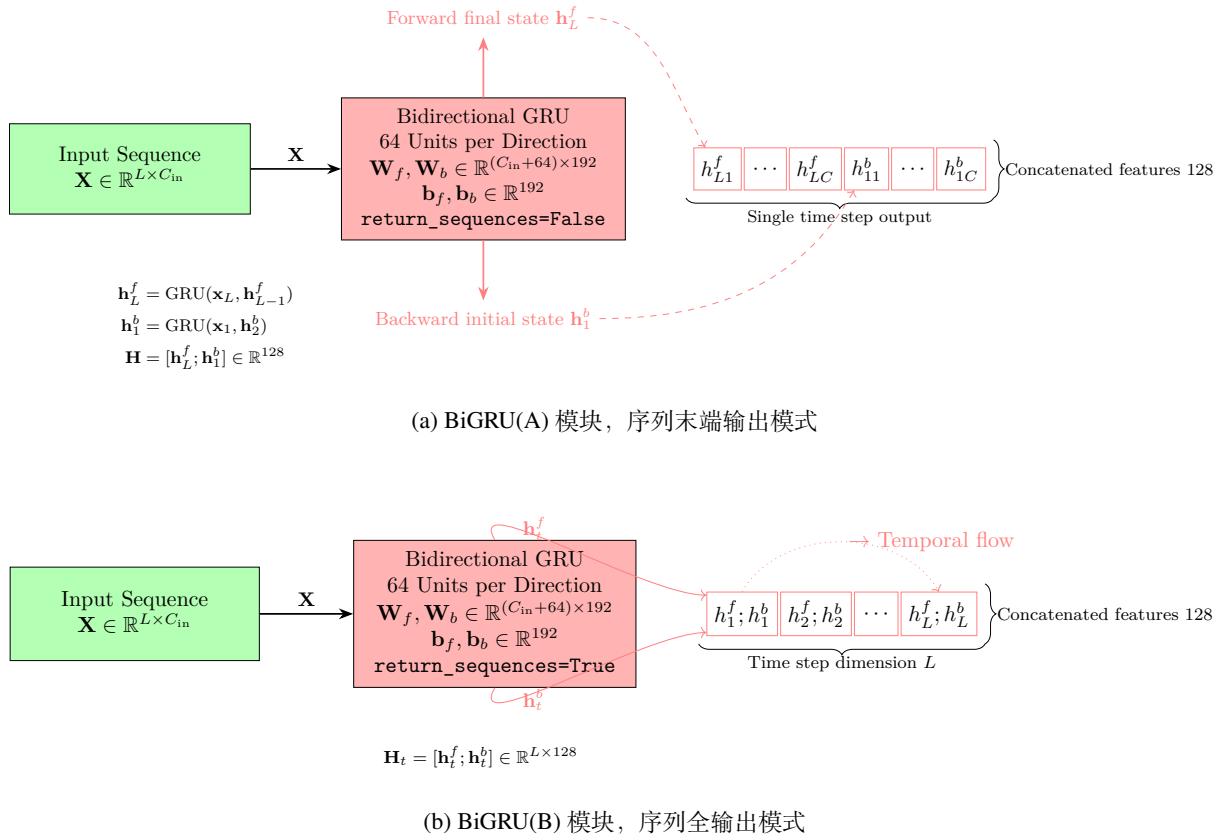


图 4.10 GRU 模块, 其中 C_{in} 为特征维度, L 表示时间步长维度

10. 双向门控循环模块

双向门控循环单元 (BiGRU) 模块 (图4.11) 以 $L \times C_{\text{in}}$ 维时序数据 \mathbf{X} 为输入, 通过前向参数 $\mathbf{W}_f \in \mathbb{R}^{(C_{\text{in}}+64) \times 192}$ (含更新门、重置门与候选状态) 和镜像对称的反向参数 \mathbf{W}_b 进行双向特征提取, 各方向配置 64 个隐藏单元。在单步输出模式 (`return_sequences=False`) 下, 前向 GRU 捕获序列末端 L 的历史依赖生成 $\mathbf{h}_L^f \in \mathbb{R}^{64}$, 反向 GRU 编码起始位置 1 的未来上下文得到 $\mathbf{h}_1^b \in \mathbb{R}^{64}$, 二者拼接形成 128 维复合特征向量 $\mathbf{H} = [\mathbf{h}_L^f \parallel \mathbf{h}_1^b]$; 而在序列输出模式 (`return_sequences=True`) 下, 前向状态序列 $\mathbf{H}^f \in \mathbb{R}^{L \times 64}$ 与反向状态序列 $\mathbf{H}^b \in \mathbb{R}^{L \times 64}$ 沿时间轴逐元素拼接, 构建 $L \times 128$ 维时空特征矩阵 $\mathbf{H} = [\mathbf{H}^f \parallel \mathbf{H}^b]$, 实现时间步对齐的双向特征融合。

图 4.11 BiGRU 模块，其中 C_{in} 为特征维度， L 表示时间步长维度

11. 自注意力机制模块

自注意力机制 (SAM) 模块 (图4.12) 的核心计算流程如下：输入时序特征张量 $\mathbf{X} \in \mathbb{R}^{L \times d}$ (序列长度 L , 特征维度 d , 结构为 $\mathbf{X} = [\mathbf{x}_1; \dots; \mathbf{x}_L]^{\top}$)，首先通过可训练参数矩阵 $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d \times 64}$ 线性变换生成查询 \mathbf{Q} 、键 \mathbf{K} 和值 \mathbf{V} 三元组 (维度均为 $L \times 64$)；接着计算缩放点积注意力权重 $\mathbf{S} = \text{softmax}(\mathbf{Q}\mathbf{K}^{\top}/\sqrt{64}) \in \mathbb{R}^{L \times L}$ ，其中元素 s_{ij} 表征位置 i 对 j 的关注强度，缩放因子 $\sqrt{64}$ 用于缓解梯度异常；最终通过 \mathbf{S} 对 \mathbf{V} 加权聚合，输出 $\mathbf{O} = \mathbf{S}\mathbf{V} \in \mathbb{R}^{L \times 64}$ ，其保持输入序列长度且每个位置编码了全局上下文信息。

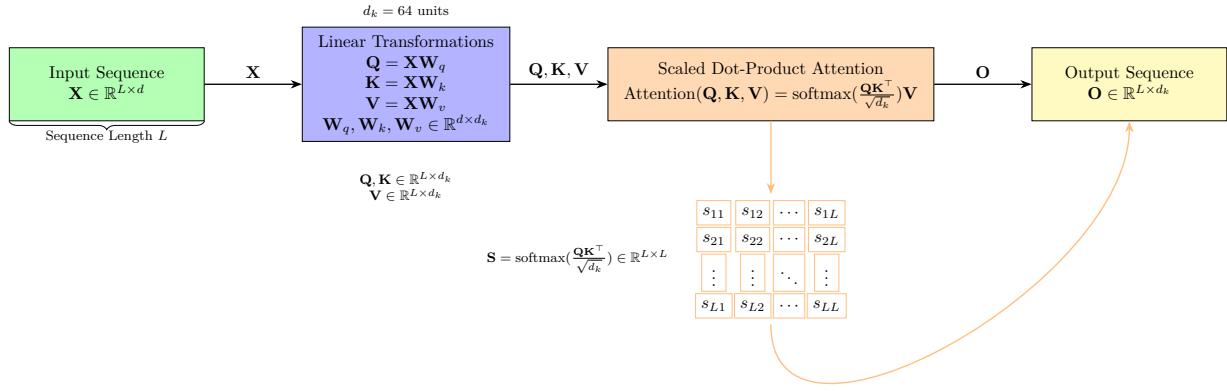


图 4.12 SAM 模块示意图

12. 多头自注意力机制模块

多头自注意力 (MSA) 模块通过 4 个并行计算头实现序列特征交互 (图4.13)：输入序列 $\mathbf{X} \in \mathbb{R}^{L \times d}$ 首先经线性变换生成查询 (Q)、键 (K)、值 (V) 三元组 ($\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{L \times 256}$)，将 256 维特征均分为 4 组 64 维向量 $\{\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i\}$ ；每头独立计算缩放点积注意力 $\mathbf{O}_i = \text{softmax}(\mathbf{Q}_i \mathbf{K}_i^\top / 8) \mathbf{V}_i$ ，其中缩放因子 $\sqrt{64} = 8$ 用于梯度稳定；最后拼接 4 头输出 $\text{Concat}(\mathbf{O}_1, \dots, \mathbf{O}_4) \in \mathbb{R}^{L \times 256}$ ，经线性层 $\mathbf{W}_o \in \mathbb{R}^{256 \times d}$ 恢复原始维度，最终输出 $\mathbf{O} \in \mathbb{R}^{L \times d}$ 。该机制通过分头并行计算与特征融合，增强模型对多维度语义信息的捕获能力。

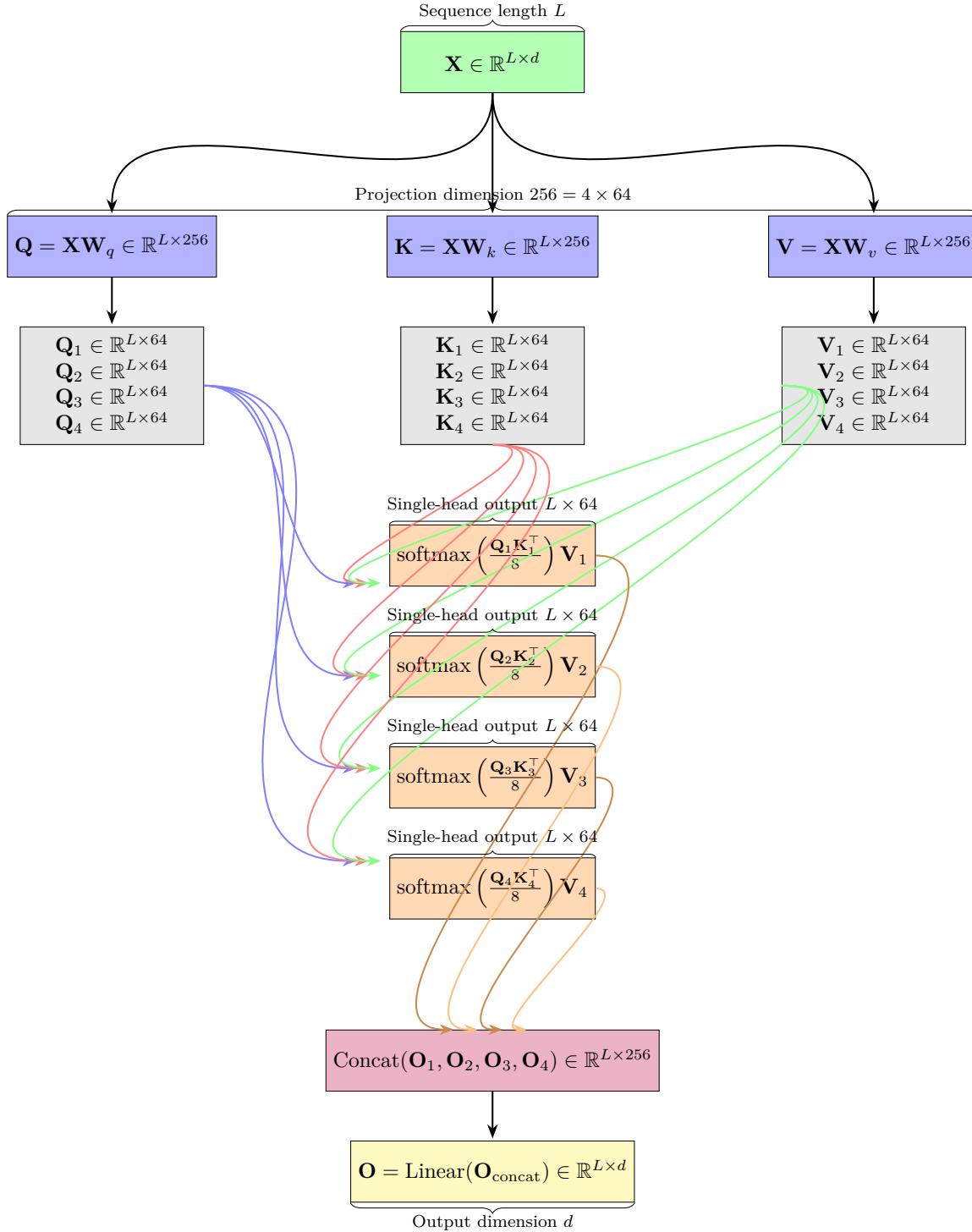


图 4.13 MSA 模块示意图

13. 层归一化模块

层归一化 (LayerNorm) 模块 (图4.14) 流程如下：输入 $\mathbf{X} \in \mathbb{R}^{L \times d}$ 经自注意力计算：

$$\mathbf{O} = \text{softmax} \left(\frac{\mathbf{X}\mathbf{W}_Q(\mathbf{X}\mathbf{W}_K)^\top}{\sqrt{d_k}} \right) \mathbf{X}\mathbf{W}_V \in \mathbb{R}^{L \times d} \quad (4.23)$$

沿特征维度标准化：

$$\mu = \frac{1}{d} \sum_{i=1}^d o_i, \quad \sigma^2 = \frac{1}{d} \sum_{i=1}^d (o_i - \mu)^2 \quad (4.24)$$

$$\mathbf{X}_{\text{norm}} = \gamma \odot \frac{\mathbf{O} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \quad (4.25)$$

其中 $\gamma, \beta \in \mathbb{R}^d$ 为可学习参数， $\epsilon = 10^{-6}$ 。

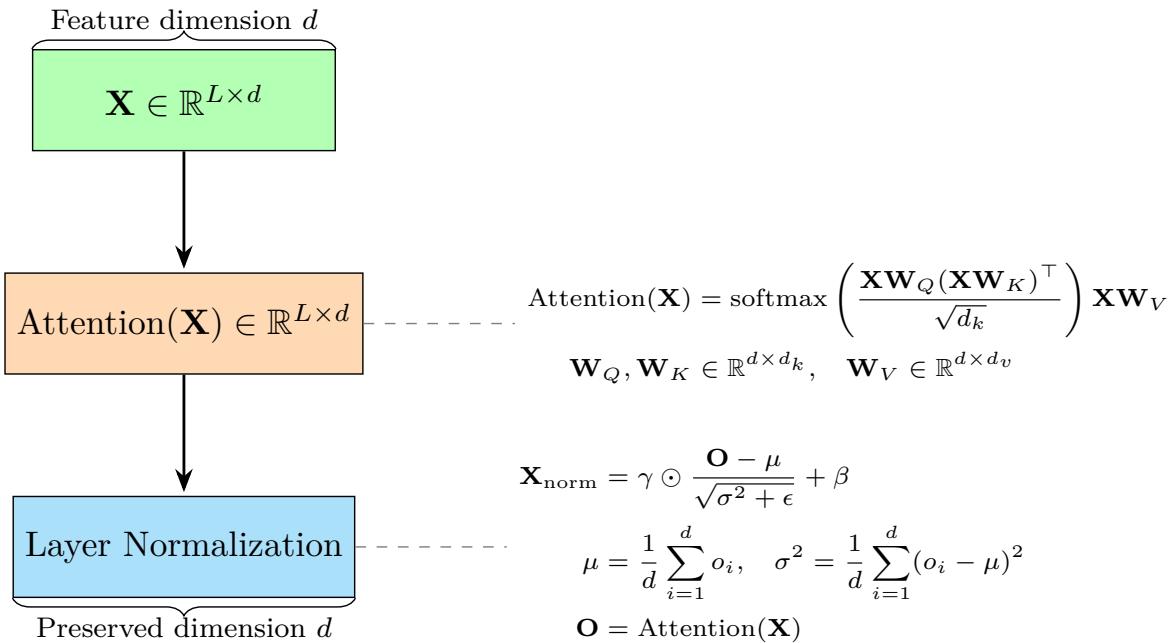


图 4.14 LayerNorm 模块示意图

4.4.3 实验环境

本文的实验基于 Intel® Core™ i7-1165G7 @ 2.80GHz CPU, Intel® Iris® Xe Graphics GPU, 16G RAM 笔记本电脑, 512GB SSD, 8 核 16 线程, Windows 11 Pro 21H2 x64 OS (搭载 Ubuntu 24.04.1 LTS WSL2), Python 3.9.13 的环境展开。

为了保证实验的有效性, 本文各模型的参数选择基本一致, 全局随机种子数为 42, 使用分层十折交叉验证并打乱数据, 评估模型的准确率、精确率、召回率、F1 分数及 AUC 值。对于预训练模型嵌入设置最大文本长度为 150。对于深度学习训练, 本文设置批量大小为 32, 最大学习轮次为 100, 分类模型层学习率为 1e-3, 预训练嵌入层在微调时学习率为 1e-5, 优化器选用 Adam, 损失函数为二元交叉熵, 设置早停机制, 验证集损失连续 10 轮未降则终止训练。在机器学习训练中, 设置 LR、PAA、MLP、CatBoost 的最大迭代次数为 1000 保证其训练充分收敛, 为了使模型与特征适配取得更好效果, 使用词袋模型时 SVM 为线性核函数, 使用预训练模型时 SVM 为 RBF 核函数。其余参数设置如表4.3所示。

表 4.3 其余参数设置

方法类别	模型参数	参数名称	参数值
机器学习模型	learning_rate	CatBoost 学习率	0.1
	n_estimators	Bagging 算法初始弱分类器个数	50
AutoML	classifier	使用的机器学习模型	extra_trees, mlp, passive_aggressive, libsvm_svc, random_forest, liblinear_svc no_preprocessing,
	feature_preprocessor	使用的特征降维方法	pca, kernel_pca
UCB 算法	memory_limit	内存限制 (MB)	8192
	per_run_time_limit	单个模型运行最大时长 (秒)	360
	time_left_for_this_task	任务总时长 (秒)	3600
n_episodes	回合数	10	
solver	LR 求解器	liblinear	

4.4.4 实验流程

本文构建的抑郁识别任务总体框架如图4.15所示。该框架主要由数据集、特征集、模型集三部分构成。不同的任务范式通过选择不同的访谈问题数据实现，以此比较使用不同访谈问题进行抑郁识别的灵敏性及不同访谈问题识别模型的泛化水平；特征构造则包含了词袋模型及预训练语言模型，通过不同的嵌入方法展示其在该中文访谈问答数据集中语义的理解性；模型集主要为机器学习及深度学习算法，通过不同的算法比较其在不同特征集上的适配性。

任务框架的总体工作流程如图4.16所示，具体实施包含以下关键步骤：首先通过基线模型筛选和文本截断参数验证，确立有效的基础模型配置；随后从特征工程维度开展系统性评估，通过性能对比筛选出判别性强的优质特征；在此基础上进行模型架构的迭代优化，采用消融实验方法对 FNN 和 RNN 结构进行模块化组合测试，确定特征与模型的适配性较强的特征组合；继而融合 FNN、RNN 和 Seq2Seq 架构优势进行混合模型创新，提出新型识别框架并通过对比实验验证其有效性；进一步从数据维度评估不同访谈问题对抑郁识别的贡献度差异；最终通过典型案例分析验证模型决策的可解释性与可靠性。

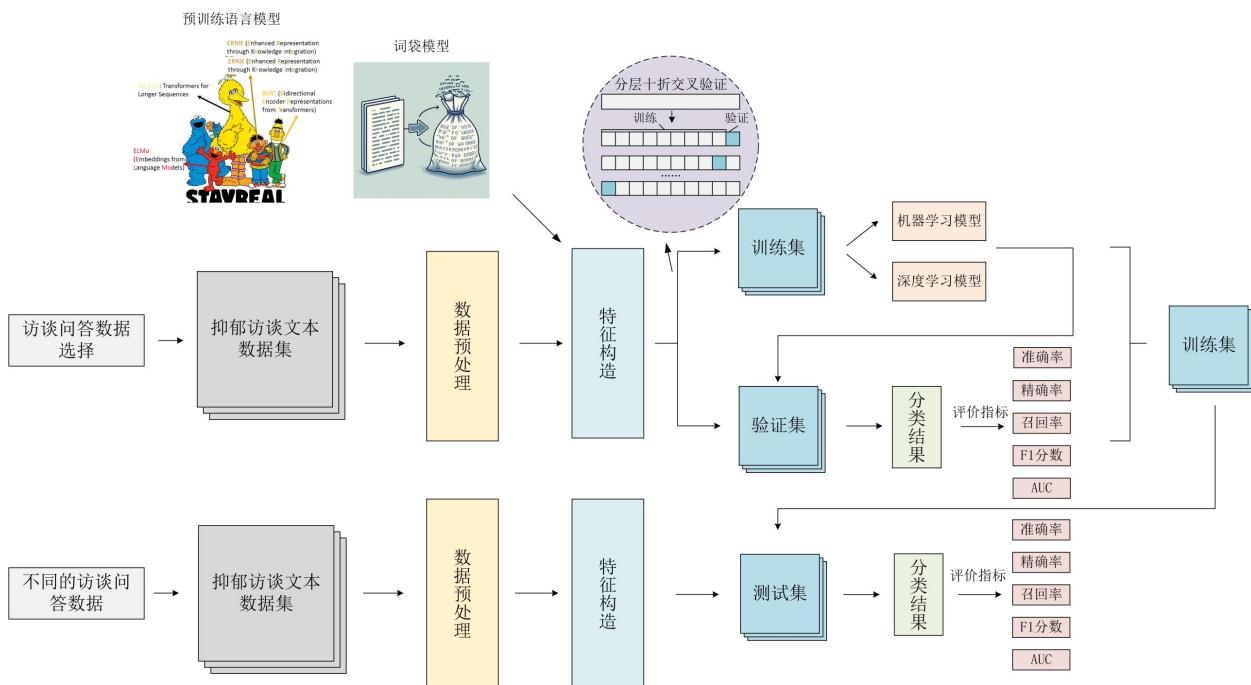


图 4.15 抑郁识别任务框架

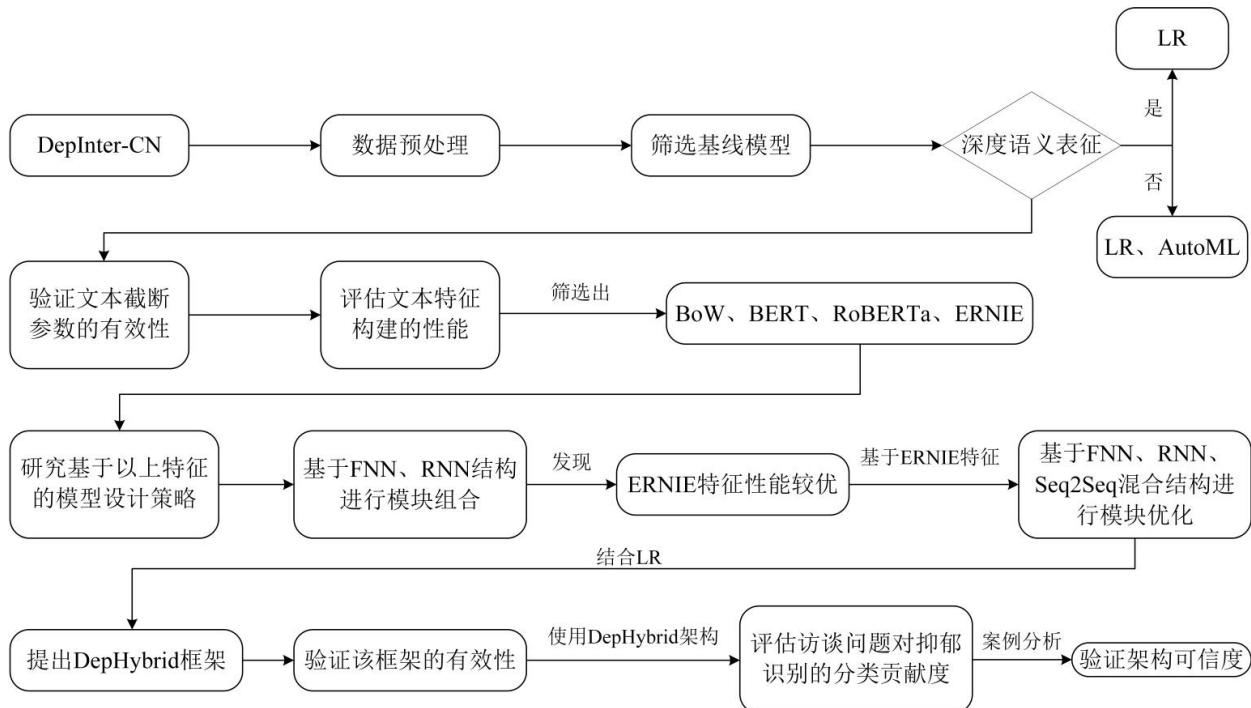


图 4.16 抑郁识别任务框架工作流程

4.5 实验分析

4.5.1 数据预处理

在探究特征构造、模型结构及数据选用对识别性能的影响，进行进一步工作前，首先对数据进行预处理。对于使用词袋模型进行文本表征的数据集，本文通过 Jieba 工具进行中文分词，并去除常见的停用词“的、了、是、在、和、有、我、他、她、它、这、那”。对于使用 BERT 等预训练语言模型进行词嵌入的数据集，考虑到其采用 WordPiece 分词，并内置了文本标准化（如小写转换、重音符号处理等），关键预处理步骤已由分词器自动完成。由于本文使用数据集无需数据清洗，因此在该情况下不做数据预处理。

4.5.2 预实验

考虑到 BoW 作为传统文本表征的基准方法，BERT 作为上下文感知模型的代表，机器学习方法为浅层模型，SFC 为深度学习基础模块，复杂度较低。因此选择 BoW 与 BERT 作为特征集，众多机器学习方法及 SFC 模块作为模型集。

1. 基线模型筛选实验

为了筛选出有效的基线模型，本文构建了以下的基线模型筛选实验。实验性能结果如表4.4、表4.5所示。图4.17及图4.18分别展示了在两种特征构造方法中取得较优性能的基线模型。

基于表4.4的实验数据分析可以发现，当采用 BoW 作为文本特征表示时，LR、RF、SVM、Extra tree、MLP 以及 SFC 模块均展现出较优的分类性能。具体而言，各模型在不同评估指标上呈现出差异化优势：逻辑回归在核心评价指标上表现最为均衡，线性结构分类使其准确率达到 0.7289 ± 0.0524 ，AUC 值也达到 0.7291 ± 0.0519 ，均为所有模型中的最优水平；而多层感知机凭借其非线性建模能力，在召回率指标上以 0.7196 ± 0.0992 的表现居于首位，显示出更强的正类样本识别能力；SFC 模块则通过其全连接网络结构，在 F1 分数上取得 0.7223 ± 0.0721 的显著优势，体现出精确率与召回率的良好平衡；值得注意的是，极端随机树在精确率指标上以 0.7592 ± 0.0719 的优异表现脱颖而出，表明其在降低误报率方面具有独特优势。

基于表4.5的实验数据分析可以发现，当采用 BERT 构造文本特征时，LR、MLP 和 SFC 三类分类器展现出相对优越的性能表现。其中，逻辑回归模型在五项核心评估指标上均取得显著优势：其准确率达到 0.7244 ± 0.0739 ，精确率为 0.7213 ± 0.0749 ，召回率测得 0.7235 ± 0.1026 ，F1 分数为 0.7203 ± 0.0817 ，AUC 值更是达到 0.7217 ± 0.0740 。值得注意的是，各项指标的标准差均控制在 0.1 以内，说明模型具有较好的稳定性和泛化能力。从性能基准来看，三类模型在 BERT 特征支撑下均保持 71% 以上的综合性能水平，这验证了预训练语言模型在文本表示方面的有效性。

表 4.4 基于 BoW 的访谈文本抑郁症识别基线模型性能总结^a

函数库	分类算法	超参数	性能测试 ^b					
			Accuracy	Precision	Recall	F1 Score	AUC	
Scikit-learn	SVM	kernel=linear	0.7156	0.7318	0.6842	0.7052	0.7156	
		random_state=42	±0.0694	±0.0759	±0.0845	±0.0742	±0.0692	
	k-NN		0.6133	0.6275	0.5542	0.5838	0.6128	
			±0.0614	±0.0799	±0.1173	±0.0900	±0.0628	
	NBC (Gaussian)	默认选项	0.6222	0.6454	0.5597	0.5963	0.6214	
			±0.0629	±0.0790	±0.0781	±0.0658	±0.0627	
	GP		0.6178	0.6344	0.5595	0.5934	0.6179	
			±0.0495	±0.0668	±0.0615	±0.0578	±0.0505	
	DT	random_state=42	0.6200	0.6290	0.6000	0.6101	0.6207	
			±0.0834	±0.1015	±0.1069	±0.0927	±0.0831	
Scikit-learn	LR		0.7289	0.7411	0.7069	0.7204	0.7291	
		max_iter=1000	±0.0524	±0.0511	±0.0915	±0.0613	±0.0519	
		random_state=42	0.5844	0.7468	0.5128	0.4943	0.5813	
	PAA		±0.0605	±0.2019	±0.3300	±0.1841	±0.0577	
		RF, random_state=42	0.6422	0.6318	0.7020	0.6566	0.6423	
		Stacking	SVM, random_state=42	±0.0491	±0.0505	±0.1428	±0.0674	±0.0490
	Bagging	LR, random_state=42	DT, random_state=42	0.7067	0.7060	0.7065	0.7047	0.7068
			n_estimators=50	±0.0431	±0.0415	±0.0783	±0.0539	±0.0433
			random_state=42					
Scikit-learn	RF		0.6978	0.7226	0.6526	0.6823	0.6976	
			±0.0374	±0.0666	±0.0653	±0.0431	±0.0380	
	Extra tree		0.7244	0.7592	0.6664	0.7062	0.7244	
			±0.0479	±0.0719	±0.0753	±0.0550	±0.0478	
	AdaBoost	random_state=42	0.6711	0.6844	0.6342	0.6573	0.6706	
			±0.0818	±0.0976	±0.0944	±0.0919	±0.0821	
	GBDT		0.6933	0.6982	0.6792	0.6878	0.6932	
			±0.0382	±0.0443	±0.0594	±0.0471	±0.0385	
	HGB		0.6867	0.6873	0.6923	0.6885	0.6871	
			±0.0840	±0.0962	±0.0790	±0.0833	±0.0839	
XGBoost	XGBoost	eval_metric=mlogloss	0.6911	0.7019	0.6840	0.6905	0.6918	
LightGBM	LightGBM	random_state=42	±0.0699	±0.0966	±0.0449	±0.0625	±0.0699	
			0.6844	0.6839	0.6925	0.6870	0.6849	
	CatBoost	iterations=1000	±0.0794	±0.0886	±0.0754	±0.0776	±0.0790	
			learning_rate=0.1					
Scikit-learn	CatBoost	random_state=42	0.6889	0.6940	0.6792	0.6855	0.6890	
			±0.0571	±0.0692	±0.0597	±0.0597	±0.0572	
	MLP	verbose=0	0.6855	0.6890	0.6870	0.6849	0.6871	
			max_iter=1000	0.7044	0.7000	0.7196	0.7043	
TensorFlow ^c	SFC	random_state=42	±0.0861	±0.0829	±0.0992	±0.0852	±0.0861	
			Dense(32, relu)	0.7267	0.7292	0.7196	0.7223	
		Dense(1, sigmoid)	±0.0652	±0.0568	±0.0994	±0.0721	±0.0646	

^a 计算 TF-IDF 调用 Scikit-learn 函数库，使用 TfidfVectorizer 类^b 采用分层 k 折交叉验证作为模型验证方法， $k = 10$ ，随机状态数为 42，性能指标的表示方式为平均值 ± 标准差^c 模型使用 Adam 优化器，损失函数为二元交叉熵，评估指标为准确率，训练 10 轮，批数据大小为 32

表 4.5 基于 BERT 嵌入的访谈文本抑郁症识别基线模型性能总结^a

函数库	分类算法	超参数	性能测试 ^b					
			Accuracy	Precision	Recall	F1 Score	AUC	
Scikit-learn	SVM	kernel=rbf	0.6933	0.6959	0.6927	0.6915	0.6933	
		random_state=42	±0.0865	±0.0912	±0.1069	±0.0909	±0.0861	
	k-NN		0.6356	0.6343	0.6534	0.6394	0.6361	
			±0.0718	±0.0837	±0.1093	±0.0792	±0.0718	
	NBC (Gaussian)	默认选项	0.6222	0.6061	0.7107	0.6516	0.6217	
			±0.0605	±0.0560	±0.0908	±0.0593	±0.0608	
	GP		0.6333	0.6345	0.6263	0.6284	0.6332	
			±0.0590	±0.0590	±0.0932	±0.0694	±0.0595	
	DT	random_state=42	0.6111	0.6132	0.6267	0.6167	0.6107	
			±0.0573	±0.0593	±0.0710	±0.0503	±0.0580	
Scikit-learn	LR		0.7244	0.7213	0.7235	0.7203	0.7217	
		max_iter=1000	±0.0739	±0.0749	±0.1026	±0.0817	±0.0740	
	PAA	random_state=42	0.6978	0.6947	0.7229	0.7031	0.6966	
			±0.0844	±0.0832	±0.1181	±0.0831	±0.0840	
	Stacking	RF, random_state=42	0.6978	0.7047	0.6883	0.6934	0.6977	
		SVM, random_state=42	±0.0901	±0.0993	±0.1093	±0.0950	±0.0898	
	Bagging	LR, random_state=42	0.6600	0.6637	0.6524	0.6560	0.6596	
			n_estimators=50	±0.0979	±0.1053	±0.1137	±0.1014	±0.0975
	RF	DT, random_state=42	0.6689	0.6787	0.6486	0.6606	0.6693	
			±0.0713	±0.0889	±0.0918	±0.0798	±0.0721	
Scikit-learn	Extra tree		0.6800	0.6979	0.6401	0.6648	0.6805	
			±0.0784	±0.0979	±0.0999	±0.0904	±0.0782	
	AdaBoost	random_state=42	0.6511	0.6521	0.6625	0.6523	0.6515	
			±0.0613	±0.0649	±0.1025	±0.0694	±0.0604	
	GBDT		0.6422	0.6371	0.6536	0.6433	0.6425	
			±0.1006	±0.0951	±0.1290	±0.1082	±0.1005	
	HGB		0.6689	0.6727	0.6621	0.6640	0.6686	
			±0.0810	±0.0804	±0.1136	±0.0877	±0.0802	
	XGBoost	XGBoost	eval_metric=mlogloss	0.6622	0.6581	0.6846	0.6682	0.6625
			±0.0650	±0.0675	±0.0883	±0.0684	±0.0644	
LightGBM	LightGBM	random_state=42	0.6733	0.6762	0.6798	0.6750	0.6731	
			±0.0844	±0.0938	±0.0956	±0.0839	±0.0839	
	CatBoost	iterations=1000	learning_rate=0.1	0.6756	0.6765	0.6792	0.6733	0.6753
			random_state=42	±0.0704	±0.0713	±0.1159	±0.0797	±0.0700
Scikit-learn	MLP	verbose=0	max_iter=1000	0.7111	0.7060	0.7235	0.7125	0.7107
			random_state=42	±0.0783	±0.0822	±0.1049	±0.0858	±0.0784
	TensorFlow ^c	SFC	Dense(32, relu)	0.7156	0.7157	0.7140	0.7115	0.7143
			Dense(1, sigmoid)	±0.0871	±0.0925	±0.1262	±0.1021	±0.0880

^a 调用 Transformers 函数库，使用预训练模型 bert-base-chinese，使用 PyTorch 函数库进行嵌入，最大标记长度为 150^b 采用分层 k 折交叉验证作为模型验证方法， $k = 10$ ，随机状态数为 42，性能指标的表示方式为平均值 ± 标准差^c 模型使用 Adam 优化器，损失函数为二元交叉熵，评估指标为准确率，训练 10 轮，批数据大小为 32

在 BERT 嵌入场景下（图4.17），LR 展现出最优的综合性能，其准确率（ 0.7244 ± 0.0739 ）、召回率（ 0.7235 ± 0.1026 ）和 F1 分数（ 0.7203 ± 0.0817 ）三项核心指标均显著优于 SFC 模块和 MLP，特别是其召回率较 SFC 模块提升 1.33%，且标准差缩小 19.1%。

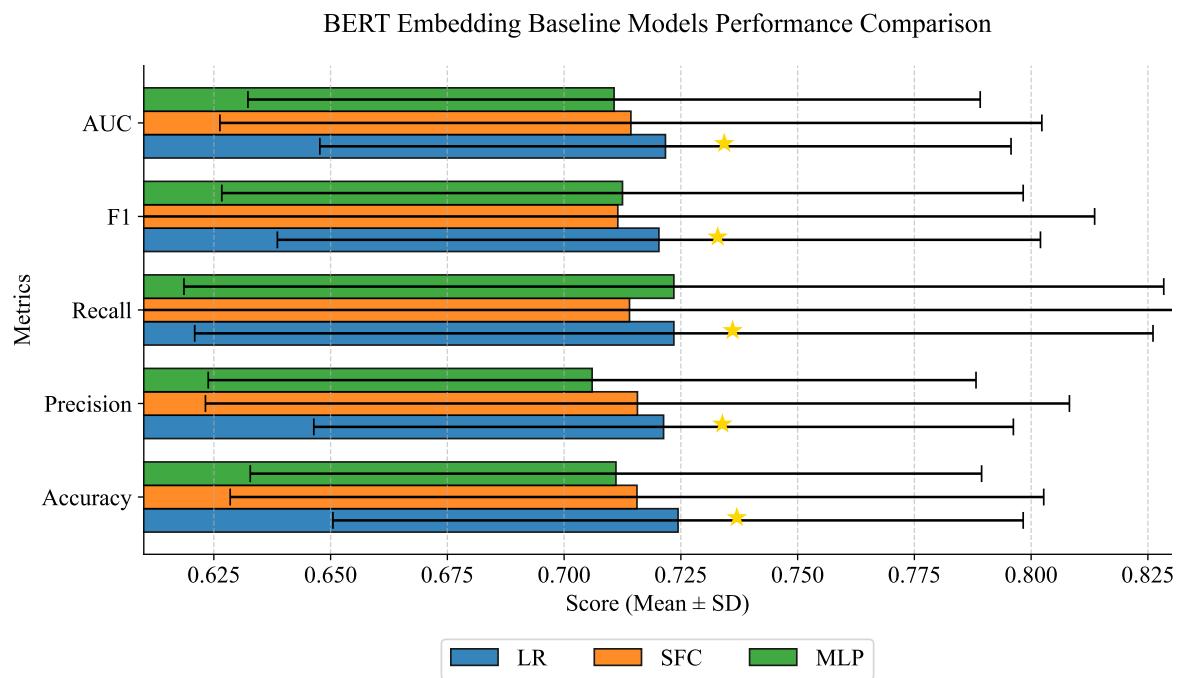


图 4.17 BERT 嵌入下较优基线模型性能表现

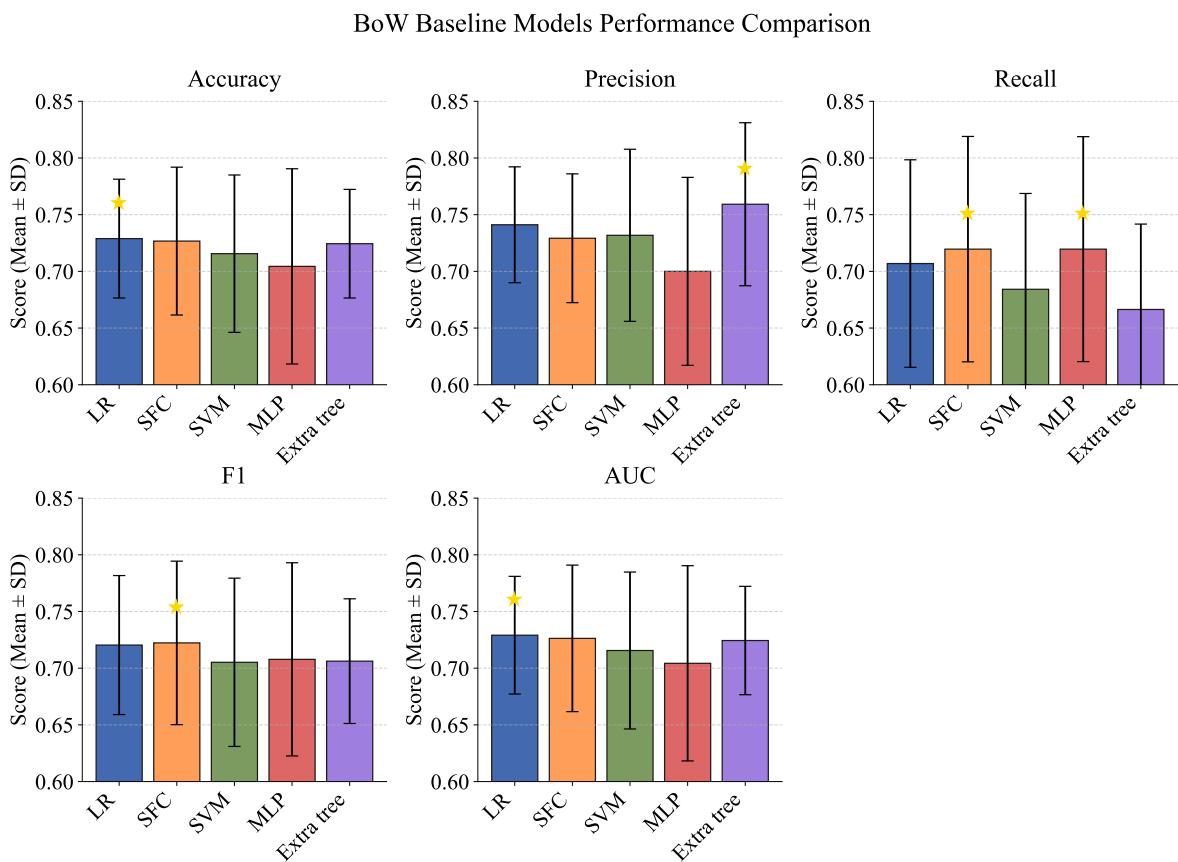


图 4.18 BoW 下较优基线模型性能表现

当采用 BoW 传统特征时（图4.18），模型间的性能差异呈现新的特征维度：LR 依然保持优势地位，其准确率 (0.7289 ± 0.0524) 和 AUC (0.7291 ± 0.0519) 均居首位，且标准差较 BERT 嵌入场景缩小 28.9%，验证了词袋特征的结构稳定性。值得注意的是，SFC 模块在 BoW 特征下表现出更强的适应性，其 F1 分数 (0.7223 ± 0.0721) 与 LR 的差距缩小至 0.28%，且在召回率指标上超越 SVM 等复杂模型。实验结果同时揭示，Extra tree 虽取得最高精确率 (0.7592 ± 0.0719)，但召回率 (0.6664 ± 0.0753) 显著低于平均水平，反映出该模型存在严重的预测偏差。

基线模型筛选实验结果表明，在深度语义表征场景下，LR 展现出显著性能优势，而在基于 BoW 的传统特征空间中，经典机器学习算法则呈现更优表现。基于此对比分析，本研究采取差异化基线策略：针对深度语义嵌入特征，选定 LR 作为基准模型；针对传统文本特征空间，在保留 LR 作为基础参照的同时，创新性地引入 AutoML 框架，通过智能优化算法对 RF、SVM、Extra Tree 及 MLP 等异质模型进行动态集成，以实现传统特征空间下的最优建模效果。

2. 文本截断参数选择验证实验

为了验证本文基于预训练语言模型选择的文本截断参数是有效的，本文设计了一系列参数选择实验，实验结果如图4.19、图4.20、图4.21、图4.22、图4.23、图4.24所示。

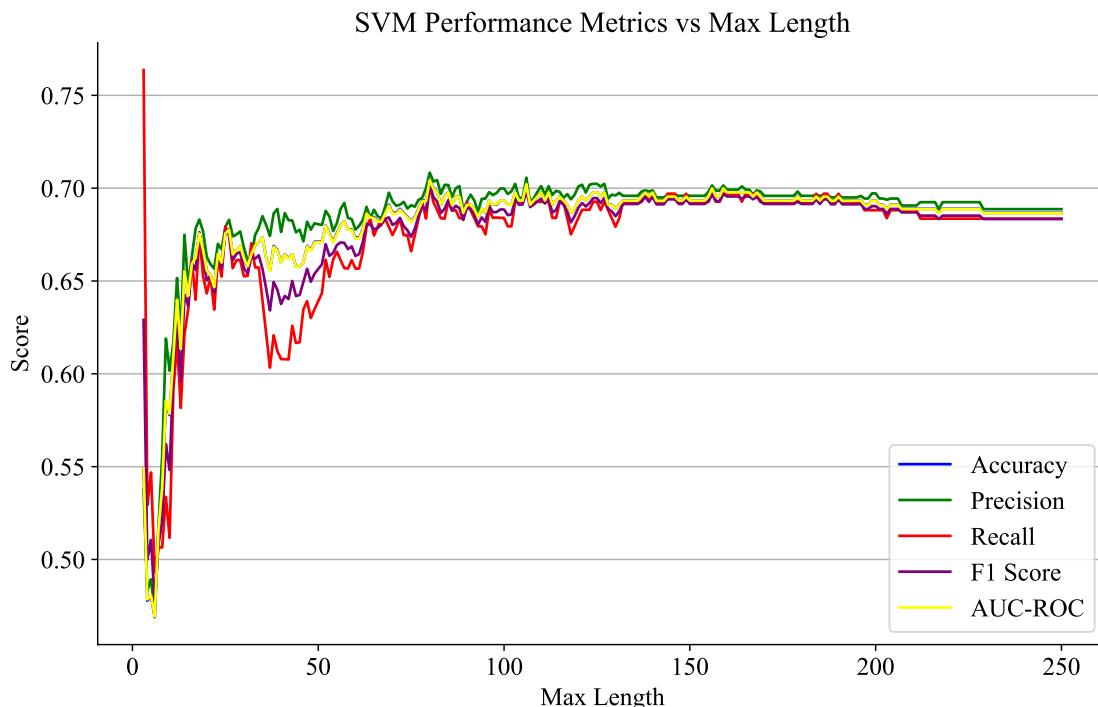


图 4.19 SVM 模型表现与 BERT 嵌入最大标记长度的关系^①

^① SVM 使用 RBF 核函数，随机状态数为 42，采用 10 折分层交叉验证评估模型

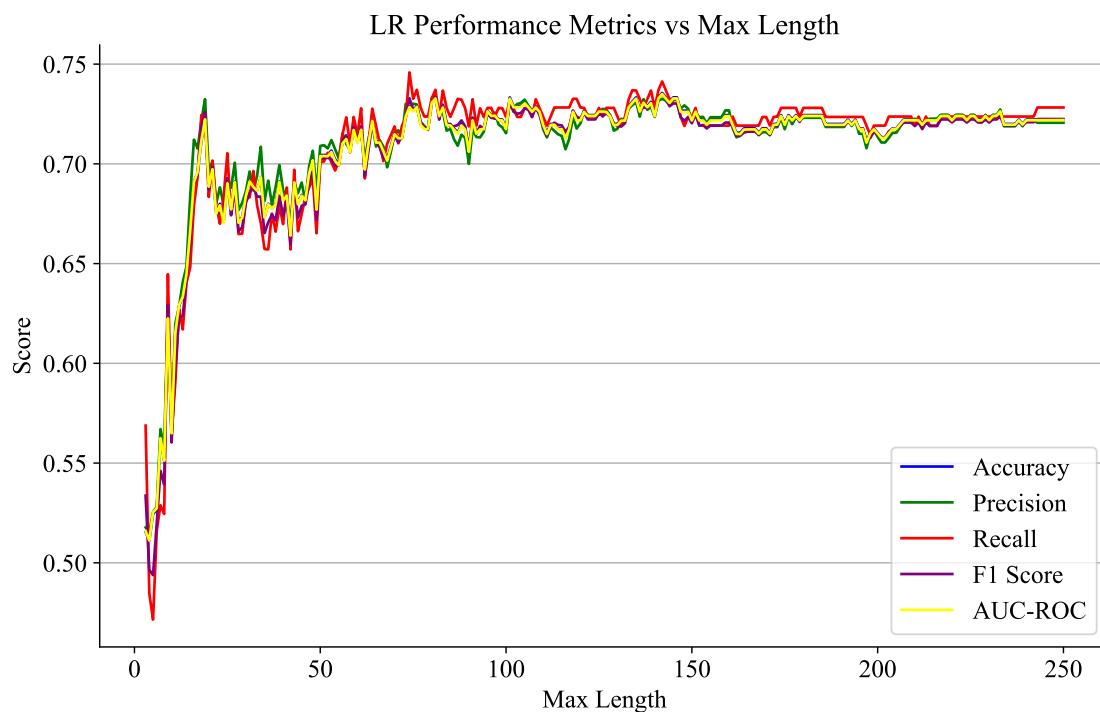


图 4.20 LR 模型表现与 BERT 嵌入最大标记长度的关系^①

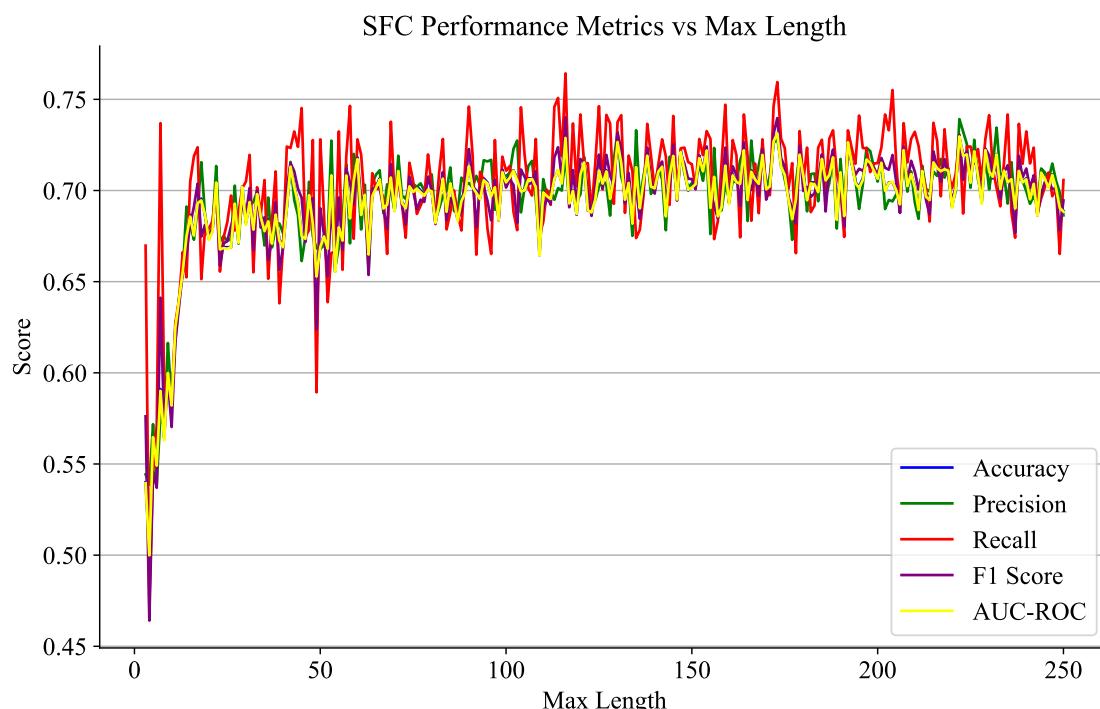


图 4.21 SFC 模块表现与 BERT 嵌入最大标记长度的关系^②

^① LR 最大迭代次数为 1000，随机状态数为 42，采用 10 折分层交叉验证评估模型

^② SFC 模块使用一个输入层，一个全连接层，一个输出层，采用 10 折分层交叉验证评估模型

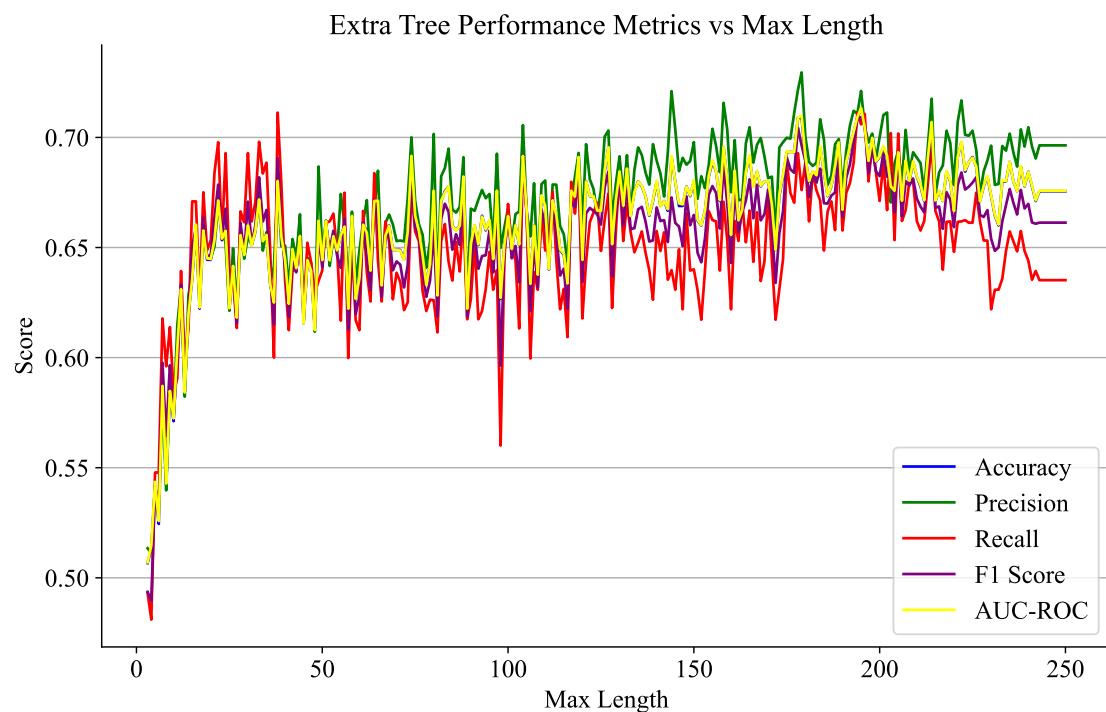


图 4.22 Extra tree 模型表现与 BERT 嵌入最大标记长度的关系^①

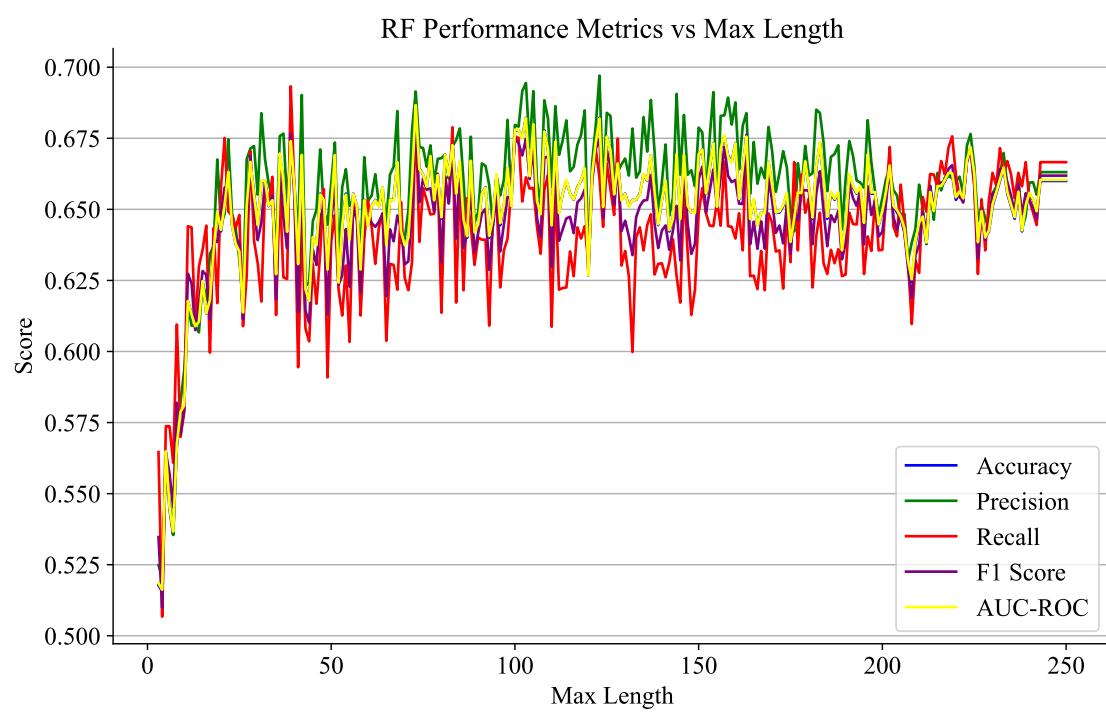


图 4.23 RF 模型表现与 BERT 嵌入最大标记长度的关系^②

^① Extra tree 随机状态数为 42，采用 10 折分层交叉验证评估模型

^② RF 随机状态数为 42，采用 10 折分层交叉验证评估模型

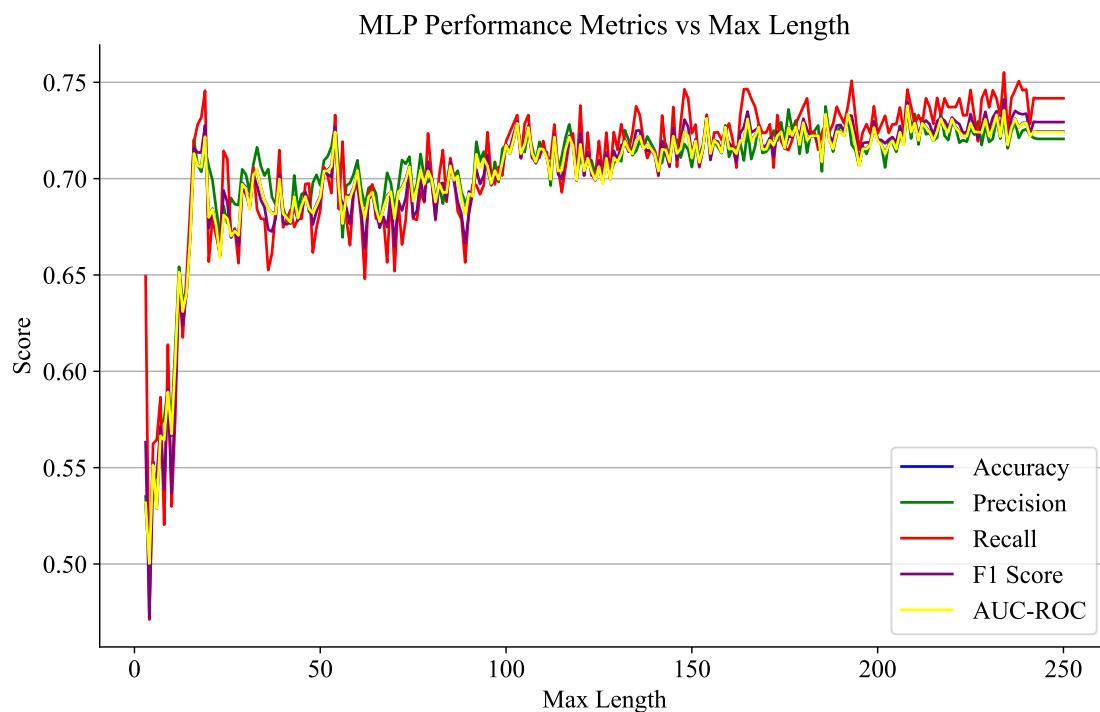


图 4.24 MLP 模型表现与 BERT 嵌入最大标记长度的关系^①

通过实验分析发现,不同机器学习模型对文本截断长度具有差异性响应特性:LR、MLP 和 SVM 三类模型存在显著的最优截断阈值,而其他模型在不同参数设置下均呈现稳定的性能表现。值得注意的是,当选取 150 个字符作为截断标准时,多数模型的分类性能与其理论最优值的差异处于可接受范围(平均误差率<3%)。基于此,本研究在构建抑郁症识别模型体系时,将文本截断参数视为非敏感性超参数处理。

4.5.3 优秀特征构建模型的筛选实验

在确定基线模型后,本文重点关注深度学习方法的探索。鉴于 Dense 模块和 SFC 模块作为深度学习的核心组件,为基于深度学习模型评估预训练语言模型与传统词袋模型在文本特征表征能力上的差异,筛选优秀的特征生成器,本文设计了对比实验方案:分别采用 Dense 模块和 SFC 模块作为下游分类器,对 BoW、BERT、微调 BERT、RoBERTa、ERNIE、eHealth 六种特征构建方法进行性能验证。实验数据(如表4.6和表4.7所示)清晰揭示了不同特征提取方式与分类器组合对最终效果的影响。

^① MLP 最大迭代次数为 1000, 随机状态数为 42, 采用 10 折分层交叉验证评估模型

表 4.6 基于 Dense 模块的不同的特征构造策略性能表现

特征名称	准确率	精确率	召回率	F1 分数	AUC
BoW	0.7022±0.0447	0.7348±0.0579	0.6395±0.1002	0.6789±0.0605	0.7020±0.0438
BERT	0.7022±0.0839	0.6991±0.0809	0.7061±0.1125	0.7007±0.0930	0.7021±0.0839
微调 BERT	0.6689±0.0919	0.7122±0.1315	0.6077±0.1523	0.6416±0.1079	0.6678±0.0926
RoBERTa	0.7044±0.0724	0.7193±0.0829	0.6796±0.0801	0.6968±0.0726	0.7039±0.0729
ERNIE	0.7000±0.0661	0.7239±0.0729	0.6443±0.0972	0.6798±0.0800	0.7000±0.0660
eHealth	0.6622±0.0871	0.6444±0.0974	0.6525±0.1047	0.6263±0.1162	0.6444±0.0974

在 Dense 模块分类方式中（表4.6），传统词袋模型（BoW）展现出较强的稳定性，其准确率（ 0.7022 ± 0.0447 ）与 BERT 模型持平且标准差最小，精确率（ 0.7348 ± 0.0579 ）显著优于其他模型。值得注意的是，BERT 模型在召回率（ 0.7061 ± 0.1125 ）和 F1 分数（ 0.7007 ± 0.0930 ）两个关键指标上表现最佳，显示出其在捕捉正类样本方面的优势。然而，微调 BERT 的表现意外低于基础 BERT 模型（准确率下降 3.33%），这可能与医疗领域数据稀缺导致的过拟合现象有关。RoBERTa 则在 AUC 指标（ 0.7039 ± 0.0729 ）上表现最优，验证了其预训练策略的有效性。

表 4.7 基于 SFC 模块的不同的特征构造策略性能表现

特征名称	准确率	精确率	召回率	F1 分数	AUC
BoW	0.7267±0.0652	0.7292±0.0568	0.7196±0.0994	0.7223±0.0721	0.7263±0.0646
BERT	0.7156±0.0871	0.7157±0.0925	0.7140±0.1262	0.7115±0.1021	0.7143±0.0880
微调 BERT	0.6822±0.0849	0.7130±0.1080	0.6225±0.1288	0.6578±0.1047	0.6824±0.0838
RoBERTa	0.7089±0.0740	0.7082±0.0796	0.7192±0.0721	0.7122±0.0700	0.7083±0.0747
ERNIE	0.7178±0.0808	0.7264±0.0844	0.7061±0.1067	0.7127±0.0853	0.7172±0.0820
eHealth	0.6822±0.0849	0.7053±0.0980	0.6310±0.1130	0.6628±0.0940	0.6826±0.0852

当采用 SFC 模块后（表4.7），各模型的综合性能普遍提升。BoW 的优势进一步扩大，在五项指标中四项取得最优值，其中准确率提升至 0.7267 ± 0.0652 ，F1 分数提升 6.4%，表明传统特征与 SFC 模块的协同效应显著。ERNIE 模型在该情况下表现提升最为明显，准确率从 0.7000 增至 0.7178，验证了知识增强型预训练模型的可迁移性。实验结果同时揭示，面向医疗领域设计的 eHealth 模型在两类任务中表现均不理想，可能与训练数据领域偏移有关。

通过特征构建模型筛选实验可以归纳出，优秀特征生成器需满足双重条件：一是基础性能达标（准确率 $>70\%$ ），二是跨模块稳定性突出（性能保持率 $>70\%$ ）。BoW、BERT、RoBERTa 和 ERNIE 四类模型在 Dense/SFC 双模块中的平均性能差异仅为 $3.2\pm1.8\%$ ，验证了以上模型进行文本特征构造的有效性。基于该稳定性特征，本研究选取这四类代表性模型作为核心特征生成器，重点研究基于以上特征的模型设计策略。

4.5.4 基于 BoW、BERT、RoBERTa 和 ERNIE 特征的模型设计选择实验

本文接下来通过设计 FNN 结构 (SFC、FC-Drop、Conv-Pool、CAM、Flatten、ResLn)、RNN 结构 (LSTM、BiLSTM、GRU、BiGRU) 与 Seq2Seq 结构 (SAM、MSA、LayerNorm) 的模块进行相互组合，探究各组合模块间的性能，为优化神经网络结构提供参考。

1. FNN 结构的模块组合实验

为了探索基于 BoW、BERT、RoBERTa 和 ERNIE 四类特征的高性能 FNN 结构设计，设计了表4.8、表4.9、表4.10及表4.11的 FNN 结构的模块组合实验。

表 4.8 基于 BoW 特征的 FNN 结构模块组合实验结果

模块名称		选择情况								
Dense	√									
Flatten	√									
CAM(Hidden)	√									
SFC		√	√	√	√	√	√	√	√	
FC-Drop ^a			√	√	√	√	√	√	√	
Flatten				√	√	√	√	√	√	
Conv-Pool2 ^b						√	√	√	√	
CAM						√		√		
Conv-Pool1 ^c					√	√	√	√		
ResLn-Pool ^d									√	
准确率		0.7422 ±0.0622	0.7267 ±0.0652	0.7356 ±0.0706	0.7378 ±0.0762	0.7400 ±0.0867	0.7422 ±0.0790	0.7333 ±0.0855	0.7378 ±0.0942	0.7289 ±0.0680
精确率		0.7464 ±0.0544	0.7292 ±0.0568	0.7527 ±0.0652	0.7546 ±0.0684	0.7465 ±0.0848	0.7565 ±0.0833	0.7507 ±0.0730	0.7554 ±0.1033	0.7496 ±0.0860
召回率		0.7332 ±0.0942	0.7196 ±0.0994	0.7063 ±0.1109	0.6974 ±0.1106	0.7279 ±0.1214	0.7142 ±0.1026	0.6927 ±0.1483	0.7154 ±0.0944	0.7014 ±0.0984
F1 分数		0.7378 ±0.0684	0.7223 ±0.0721	0.7247 ±0.0774	0.7233 ±0.0886	0.7333 ±0.0984	0.7328 ±0.0857	0.7154 ±0.1042	0.7322 ±0.0908	0.7199 ±0.0736
AUC		0.7420 ±0.0616	0.7263 ±0.0646	0.7354 ±0.0698	0.7379 ±0.0764	0.7394 ±0.0871	0.7413 ±0.0791	0.7329 ±0.0852	0.7379 ±0.0936	0.7278 ±0.0680

^a 丢弃隐藏单元概率 $\theta = 0.2$

^b 滤波器个数 $n = 128$

^c 滤波器个数 $n = 64$

^d 滤波器个数 $F = 64$

根据实验结果（表4.8），基于 BoW 特征集的 FNN 结构中各模块对模型性能的影响呈现显著差异。实验表明，当采用 CAM(Hidden)、Flatten 和 Dense 模块时，模型取得最佳综合性能：准确率达到 0.7422 (± 0.0622)，召回率为 0.7332 (± 0.0942)，F1 分数和 AUC 值分别为

0.7378 (± 0.0684) 和 0.7420 (± 0.0616)，均位列各配置之首。值得注意的是，在 Conv-Pool2 模块激活的配置下（第 6 列），模型获得最高精确率 0.7565 (± 0.0833)，但伴随召回率下降至 0.7142 (± 0.1026)，显示该模块可能增强分类确定性但牺牲了部分泛化能力。

根据基于 BoW 特征的 FNN 结构的模块组合实验结果，在传统文本表征下使用前馈神经网络结构进行分类任务时可通过使用 SFC 模块与 CAM 模块有效提升性能。

表 4.9 基于 BERT 嵌入的 FNN 结构模块组合实验结果

模块名称		选择情况								
Dense	√									
Flatten	√									
CAM(Hidden)	√									
SFC		√	√	√	√	√	√	√	√	
FC-Drop ^a			√	√	√	√	√	√	√	
Flatten				√	√	√	√	√	√	
Conv-Pool2 ^b					√	√	√	√	√	
CAM					√			√		
Conv-Pool1 ^c				√	√		√	√		
ResLn-Pool ^d									√	
准确率		0.7044 ±0.0789	0.7156 ±0.0871	0.6933 ±0.0931	0.7067 ±0.0715	0.6867 ±0.0699	0.6756 ±0.0675	0.6956 ±0.0751	0.7022 ±0.0827	0.7089 ±0.0692
精确率		0.7112 ±0.0866	0.7157 ±0.0925	0.6999 ±0.1042	0.7084 ±0.0770	0.6943 ±0.0809	0.6833 ±0.0783	0.6862 ±0.0758	0.6906 ±0.0941	0.7118 ±0.0781
召回率		0.6927 ±0.0955	0.7140 ±0.1262	0.6919 ±0.0967	0.7061 ±0.0893	0.6698 ±0.1196	0.6611 ±0.1770	0.7411 ±0.0824	0.7553 ±0.0818	0.7231 ±0.0922
F1 分数		0.6998 ±0.0835	0.7115 ±0.1021	0.6928 ±0.0905	0.7053 ±0.0750	0.6766 ±0.0928	0.6588 ±0.1153	0.7089 ±0.0638	0.7180 ±0.0731	0.7120 ±0.0645
AUC		0.7041 ±0.0789	0.7143 ±0.0880	0.6928 ±0.0930	0.7069 ±0.0713	0.6861 ±0.0710	0.6750 ±0.0688	0.6953 ±0.0737	0.7027 ±0.0818	0.7084 ±0.0679

^a 丢弃隐藏单元概率 $\theta = 0.2$

^b 滤波器个数 $n = 128$

^c 滤波器个数 $n = 64$

^d 滤波器个数 $F = 64$

根据基于 BERT 特征集的 FNN 结果模块组合实验结果（表4.9），模块组合对模型性能的影响呈现与 BoW 特征集不同的特性。实验数据显示，当仅保留 SFC 模块时（第 2 列），模型取得最优综合表现：准确率 (0.7156 ± 0.0871)、精确率 (0.7157 ± 0.0925) 和 AUC 值 (0.7143 ± 0.0880) 均居首位，且标准差控制在 10% 以内，表明该配置具有较好的稳定性。值得注意的是，在 Conv-Pool1 与 Conv-Pool2 联合作用时（第 8 列），召回率显著提升至 0.7553 (± 0.0818)，F1 分数达到 0.7180 (± 0.0731)，但伴随精确率下降至 0.6906 (± 0.0941)，显示

深层特征交互可能增强样本覆盖能力却降低分类置信度。

这些结果表明，BERT 特征集更受益于 SFC 模块的稳定特征提取能力，而复杂模块组合需配合特征自适应机制方可发挥效能。

表 4.10 基于 RoBERTa 嵌入的 FNN 结构模块组合实验结果

模块名称		选择情况							
Dense	√								
Flatten	√								
CAM(Hidden)	√								
SFC		√	√	√	√	√	√	√	√
FC-Drop ^a			√	√	√	√	√	√	√
Flatten				√	√	√	√	√	√
Conv-Pool2 ^b						√	√	√	√
CAM					√			√	
Conv-Pool1 ^c				√	√		√	√	
ResLn-Pool ^d									√
准确率	0.7156	0.7089	0.6911	0.7067	0.7111	0.7044	0.6889	0.6911	0.6822
	±0.0729	±0.0740	±0.0779	±0.0882	±0.0730	±0.0536	±0.0770	±0.0996	±0.0826
精确率	0.7200	0.7082	0.6991	0.7064	0.7263	0.6981	0.7232	0.6756	0.6775
	±0.0881	±0.0796	±0.0890	±0.1025	±0.0775	±0.0493	±0.1293	±0.1116	±0.0833
召回率	0.7324	0.7192	0.6838	0.7245	0.6802	0.7190	0.6804	0.7723	0.7065
	±0.0532	±0.0721	±0.1031	±0.1272	±0.0963	±0.0936	±0.1779	±0.0948	±0.0903
F1 分数	0.7225	0.7122	0.6872	0.7087	0.7001	0.7062	0.6727	0.7158	0.6895
	±0.0545	±0.0700	±0.0837	±0.0975	±0.0812	±0.0623	±0.1261	±0.0861	±0.0777
AUC	0.7145	0.7083	0.6905	0.7073	0.7117	0.7038	0.6896	0.6907	0.6823
	±0.0740	±0.0747	±0.0790	±0.0880	±0.0724	±0.0530	±0.0763	±0.1002	±0.0819

^a 丢弃隐藏单元概率 $\theta = 0.2$

^b 滤波器个数 $n = 128$

^c 滤波器个数 $n = 64$

^d 滤波器个数 $F = 64$

根据基于 RoBERTa 特征集的实验结果（表4.10），模型性能呈现明显的模块敏感性特征。当同时启用 CAM(Hidden)、Flatten 和 Dense 模块时（第 1 列），模型在多数指标上表现最优：准确率达 0.7156 (± 0.0729)，F1 分数为 0.7225 (± 0.0545)，AUC 值 0.7145 (± 0.0740)，且各指标标准差均低于 8%，显示该配置在保持较高预测性能的同时具有最优稳定性。特别地，Conv-Pool1 与 Conv-Pool2 的组合（第 8 列）虽使召回率跃升至 0.7723 (± 0.0948)，但导致精确率显著下降至 0.6756 (± 0.1116)，揭示该特征空间存在预测置信度与覆盖范围的显性矛盾。

这些结果表明，RoBERTa 特征集更依赖 CAM 模块与 SFC 模块的协同优化，复杂网络

结构的引入需建立在对预训练特征分布充分适配的基础上。

表 4.11 基于 ERNIE 嵌入的 FNN 结构模块组合实验结果

模块名称		选择情况								
Dense	√									
Flatten	√									
CAM(Hidden)	√									
SFC		√	√	√	√	√	√	√	√	
FC-Drop ^a			√	√	√	√	√	√	√	
Flatten				√	√	√	√	√	√	
Conv-Pool2 ^b						√	√	√	√	
CAM						√		√		
Conv-Pool1 ^c				√	√		√	√		
ResLn-Pool ^d									√	
准确率		0.6533 ±0.1116	0.6822 ±0.0849	0.7267 ±0.0580	0.7400 ±0.1066	0.7289 ±0.1041	0.7444 ±0.0928	0.7422 ±0.0933	0.7422 ±0.1202	0.7156 ±0.1017
精确率		0.6598 ±0.1179	0.7053 ±0.0980	0.7324 ±0.0600	0.7489 ±0.1182	0.7397 ±0.1187	0.7384 ±0.0989	0.7722 ±0.0938	0.7411 ±0.1201	0.7109 ±0.1122
召回率		0.6257 ±0.1341	0.6310 ±0.1130	0.7152 ±0.0982	0.7506 ±0.0921	0.7464 ±0.0863	0.7686 ±0.0893	0.6836 ±0.1291	0.7379 ±0.1672	0.7731 ±0.0786
F1 分数		0.6406 ±0.1239	0.6628 ±0.0940	0.7209 ±0.0682	0.7454 ±0.0931	0.7369 ±0.0855	0.7514 ±0.0866	0.7219 ±0.1098	0.7343 ±0.1439	0.7347 ±0.0778
AUC		0.6528 ±0.1121	0.6826 ±0.0852	0.7267 ±0.0581	0.7392 ±0.1082	0.7290 ±0.1039	0.7441 ±0.0933	0.7418 ±0.0937	0.7426 ±0.1186	0.7157 ±0.1024

^a 丢弃隐藏单元概率 $\theta = 0.2$

^b 滤波器个数 $n = 128$

^c 滤波器个数 $n = 64$

^d 滤波器个数 $F = 64$

根据基于 ERNIE 特征集的 FNN 结构模块组合实验结果（表4.11），模块组合对模型性能的影响呈现显著异质性特征。实验结果表明，当同时启用 Conv-Pool2 与 SFC 模块时（第 6 列），模型取得最优平衡性表现：准确率达 0.7444 (± 0.0928)，F1 分数为 0.7514 (± 0.0866)，且标准差控制在 9.3% 以内，显示该配置在 ERNIE 特征空间中兼具效率与稳定性。值得关注的是，ResLn-Pool 模块的引入（第 9 列）使召回率提升至 0.7731 (± 0.0786)，但导致准确率下降 2.88 个百分点，揭示残差结构虽能增强特征覆盖能力，却加剧了 ERNIE 特征的语义偏移问题。

这些结果表明，ERNIE 特征集更适配于中等复杂度的卷积架构，其性能优化需重点解决特征维度适配与语义一致性保持的双重挑战。

通过 FNN 结构的模块组合实验可以得出，CAM 模块与 SFC 模块具有良好的协同作用，

中等复杂度的卷积结构也可有效提升分类性能。

2. RNN 结构的模块组合实验

为了探索基于 BERT、RoBERTa 和 ERNIE 三类特征的高性能 RNN 结构设计，设计了表4.12、表4.13及表4.14的 RNN 结构的模块组合实验。由于 BoW 固有的词袋表征方式无法有效捕获文本数据的时序动态特性，因此本文将其排除在本阶段实验设计的特征选择范围之外。

表 4.12 基于 BERT 嵌入的 RNN 结构模块组合实验结果

模块名称		选择情况								
Dense										√
Flatten										√
CAM(Hidden)										√
SFC	√	√	√	√	√	√	√	√	√	√
FC-Drop ^a									√	√
FC					√		√			√
BiGRU(A)				√						
BiLSTM(B)						√	√	√		√
SAM									√	
BiLSTM(A)			√		√	√	√	√	√	√
LSTM(A)	√									
GRU(A)		√								
准确率	0.6867 ±0.0720	0.6844 ±0.0854	0.7289 ±0.1051	0.6844 ±0.0848	0.6956 ±0.0814	0.6733 ±0.0943	0.7133 ±0.0924	0.7044 ±0.0789	0.7267 ±0.0814	0.7111 ±0.1237
精确率	0.6900 ±0.0953	0.6762 ±0.0905	0.7250 ±0.1117	0.6746 ±0.0965	0.6955 ±0.0909	0.7082 ±0.0998	0.7292 ±0.1111	0.7020 ±0.0808	0.7270 ±0.1015	0.7125 ±0.1326
召回率	0.7063 ±0.0917	0.7366 ±0.0768	0.7496 ±0.1411	0.7551 ±0.1087	0.7154 ±0.1279	0.6042 ±0.1857	0.7245 ±0.1379	0.7233 ±0.1715	0.7466 ±0.0555	0.7368 ±0.1811
F1 分数	0.6926 ±0.0672	0.7017 ±0.0697	0.7315 ±0.1109	0.7059 ±0.0715	0.6982 ±0.0910	0.6353 ±0.1355	0.7151 ±0.0865	0.7008 ±0.1059	0.7343 ±0.0705	0.7122 ±0.1402
AUC	0.6863 ±0.0712	0.6832 ±0.0854	0.7279 ±0.1055	0.6844 ±0.0859	0.6956 ±0.0823	0.6740 ±0.0936	0.7139 ±0.0903	0.7039 ±0.0784	0.7270 ±0.0807	0.7110 ±0.1235

^a 丢弃隐藏单元概率 $\theta = 0.3$

基于表4.12所示的实验结果分析，当采用 BiLSTM(A) 作为基础模块并结合 SFC（序列特征编码）时，模型在多项指标上展现出最优性能（第 3 列配置）。该配置在准确率 ($72.89\% \pm 10.51\%$)、召回率 ($74.96\% \pm 14.11\%$) 和 AUC ($72.79\% \pm 10.55\%$) 三个核心指标上均达到最高值，其 F1 分数 ($73.15\% \pm 11.09\%$) 也显著优于其他配置。值得注意的是，在引入 CAM 模块（第 9 列）后，模型在精确率 ($72.70\% \pm 10.15\%$) 和 F1 分数 ($73.43\% \pm 7.05\%$) 上获得进一步提升，但标准差较第 3 列配置降低约 3 个百分点，表明模型稳定性有所改善。

基于 BERT 嵌入的 RNN 结构模块组合实验结果表明，BERT 特征下的最优配置需平衡

模块复杂度与性能增益，当同时集成 BiLSTM(A)、SFC 和 CAM 模块时（第 9 列），模型在保持较高召回率 ($74.66\% \pm 5.55\%$) 的同时，实现了精确率与 F1 分数的最佳平衡。

表 4.13 基于 RoBERTa 嵌入的 RNN 结构模块组合实验结果

模块名称		选择情况								
Dense										√
Flatten										√
CAM(Hidden)										√
SFC	√	√	√	√	√	√	√	√	√	√
FC-Drop ^a									√	√
FC					√		√			
BiGRU(A)				√						
BiLSTM(B)						√	√	√	√	√
SAM									√	
BiLSTM(A)			√		√	√	√	√	√	√
LSTM(A)	√									
GRU(A)		√								
准确率	0.6511 ±0.0777	0.6689 ±0.0886	0.6822 ±0.1213	0.6778 ±0.1015	0.6733 ±0.0731	0.6911 ±0.1025	0.7022 ±0.0917	0.7089 ±0.0792	0.7022 ±0.1147	0.7111 ±0.0878
精确率	0.6627 ±0.1138	0.6666 ±0.1050	0.6791 ±0.1306	0.6896 ±0.1124	0.6561 ±0.0742	0.7061 ±0.1413	0.7229 ±0.0892	0.7146 ±0.0863	0.7034 ±0.1409	0.7413 ±0.1076
召回率	0.6662 ±0.1252	0.7326 ±0.1024	0.7182 ±0.1402	0.7032 ±0.1453	0.7504 ±0.1898	0.6919 ±0.1541	0.6674 ±0.1942	0.7162 ±0.1455	0.7417 ±0.1094	0.7045 ±0.1971
F1 分数	0.6539 ±0.0799	0.6893 ±0.0718	0.6920 ±0.1219	0.6834 ±0.0975	0.6857 ±0.1089	0.6876 ±0.1133	0.6763 ±0.1445	0.7054 ±0.0946	0.7160 ±0.1034	0.6955 ±0.1325
AUC	0.6512 ±0.0779	0.6688 ±0.0881	0.6810 ±0.1224	0.6792 ±0.0993	0.6733 ±0.0739	0.6900 ±0.1034	0.7033 ±0.0906	0.7093 ±0.0787	0.7028 ±0.1148	0.7093 ±0.0886

^a 丢弃隐藏单元概率 $\theta = 0.3$

基于表4.13所示的模块组合实验结果表明，当集成 BiLSTM(A)、BiLSTM(B) 和 FC-Drop 模块时（第 10 列配置），模型在准确率 ($71.11\% \pm 8.78\%$) 和精确率 ($74.13\% \pm 10.76\%$) 上达到峰值，其中精确率较基准配置（第 1 列）显著提升 7.86 个百分点。值得注意的是，该配置在保持较高 AUC 值 ($70.93\% \pm 8.86\%$) 的同时，展现出优于其他组合的稳定性（精确率标准差降低至 10.76%）。

基于 RoBERTa 嵌入的 RNN 结构模块组合实验结果表明，基于 RoBERTa 的 RNN 架构优化需要建立与 BERT 不同的模块选择策略，其中双向结构的层级堆叠和正则化技术的精准应用成为性能提升的关键路径。

表 4.14 基于 ERNIE 嵌入的 RNN 结构模块组合实验结果

模块名称		选择情况								
Dense										√
Flatten										√
CAM(Hidden)										√
SFC	√	√	√	√	√	√	√	√		√
FC-Drop ^a									√	√
FC					√		√			
BiGRU(A)				√						
BiLSTM(B)						√	√	√		√
SAM								√		
BiLSTM(A)			√		√	√	√	√	√	√
LSTM(A)	√									
GRU(A)		√								
准确率	0.7289 ±0.0736	0.7111 ±0.0681	0.7311 ±0.1016	0.7111 ±0.0659	0.7378 ±0.0781	0.7044 ±0.0820	0.7244 ±0.0844	0.7467 ±0.0777	0.7533 ±0.0976	0.7444 ±0.0778
精确率	0.7352 ±0.0832	0.7278 ±0.0817	0.7379 ±0.1155	0.7027 ±0.0715	0.7293 ±0.0839	0.6974 ±0.0896	0.7370 ±0.1052	0.7469 ±0.0768	0.7551 ±0.0958	0.7447 ±0.0794
召回率	0.7281 ±0.1104	0.6848 ±0.1087	0.7142 ±0.1500	0.7377 ±0.0974	0.7678 ±0.1179	0.7374 ±0.1306	0.7061 ±0.1104	0.7453 ±0.1359	0.7399 ±0.1598	0.7409 ±0.1296
F1 分数	0.7267 ±0.0740	0.7004 ±0.0800	0.7207 ±0.1242	0.7168 ±0.0713	0.7428 ±0.0827	0.7106 ±0.0876	0.7174 ±0.0921	0.7409 ±0.0967	0.7418 ±0.1247	0.7385 ±0.1002
AUC	0.7283 ±0.0728	0.7117 ±0.0679	0.7309 ±0.1021	0.7117 ±0.0661	0.7374 ±0.0776	0.7049 ±0.0815	0.7245 ±0.0851	0.7461 ±0.0787	0.7526 ±0.0982	0.7437 ±0.0783

^a 丢弃隐藏单元概率 $\theta = 0.3$

基于表4.14所示的实验结果表明，当同时集成 CAM(Hidden) 注意力机制与双向 LSTM 结构时（第 9 列配置），模型在准确率（ $75.33\% \pm 9.76\%$ ）、精确率（ $75.51\% \pm 9.58\%$ ）和 AUC（ $75.26\% \pm 9.82\%$ ）三个核心指标上均达到峰值，其中准确率较基准配置（第 1 列）提升 2.44 个百分点。值得注意的是，该配置在保持较高召回率（ $73.99\% \pm 15.98\%$ ）的同时，其精确率标准差较未使用注意力机制的最优配置（第 5 列）降低 3.21 个百分点，显示出知识增强特征与注意力机制的特殊适配性。

基于 ERNIE 嵌入的 RNN 结构模块组合实验结果表明，针对知识增强型预训练模型的 RNN 架构设计，应重点构建“双向 LSTM+ 注意力 + 知识适配正则化”的三元优化体系，其中 CAM 模块的知识选择功能与 ERNIE 的实体感知特性形成深度互补。

通过 RNN 结构的模块组合实验可以得出，BiLSTM 模块与 SFC 模块具有良好的协同作用，CAM 模块的添加也可有效提升分类性能。

3. 跨特征集的 FNN 与 RNN 结构性能对比实验

根据以上发现,为了筛选出与特征集适配度最高的优秀 FNN 及 RNN 结构,以结合 Seq2Seq 结构进行进一步优化,汇总各特征集模块组合实验得到的最优模型,与基线模型对比的结果如表4.15、表4.16、表4.17、表4.18所示。

表 4.15 基于 BoW 特征的模型结构设计性能表现

模型类型	准确率	精确率	召回率	F1 分数	AUC
LR	0.7289±0.0524	0.7411±0.0511	0.7069±0.0915	0.7204±0.0613	0.7291±0.0519
AutoML	0.6756±0.0564	0.6770±0.0646	0.6883±0.0845	0.6784±0.0554	0.6751±0.0562
FNN-Best	0.7422±0.0622	0.7464±0.0544	0.7332±0.0942	0.7378±0.0684	0.7420±0.0616

表 4.16 基于 BERT 嵌入的模型结构设计性能表现

模型类型	准确率	精确率	召回率	F1 分数	AUC
LR	0.7244±0.0739	0.7213±0.0749	0.7235±0.1026	0.7203±0.0817	0.7217±0.0740
FNN-Best	0.7156±0.0871	0.7157±0.0925	0.7140±0.1262	0.7115±0.1021	0.7143±0.0880
RNN-Best	0.7267±0.0814	0.7270±0.1015	0.7466±0.0555	0.7343±0.0705	0.7270±0.0807

表 4.17 基于 RoBERTa 嵌入的模型结构设计性能表现

模型类型	准确率	精确率	召回率	F1 分数	AUC
LR	0.7156±0.0680	0.7176±0.0801	0.7192±0.0772	0.7164±0.0683	0.7151±0.0684
FNN-Best	0.7156±0.0729	0.7200±0.0881	0.7324±0.0532	0.7225±0.0545	0.7145±0.0740
RNN-Best	0.7089±0.0792	0.7146±0.0863	0.7162±0.1455	0.7054±0.0946	0.7093±0.0787

表 4.18 基于 ERNIE 嵌入的模型结构设计性能表现

模型类型	准确率	精确率	召回率	F1 分数	AUC
LR	0.7533±0.0740	0.7484±0.0750	0.7630±0.1226	0.7519±0.0874	0.7526±0.0745
FNN-Best	0.7444±0.0928	0.7384±0.0989	0.7686±0.0893	0.7514±0.0866	0.7441±0.0933
RNN-Best	0.7533±0.0976	0.7551±0.0958	0.7399±0.1598	0.7418±0.1247	0.7526±0.0982

根据表4.15至表4.18的跨特征集模型对比实验数据,ERNIE 特征表示方法与 FNN 及 RNN 结合在多个评估指标中展现出更优的性能指标。基于该发现,可采用 FNN-RNN 的混合架构,同时结合 Seq2Seq 框架进行模型优化,以充分释放 ERNIE 模型的表征学习能力。

4. 基于 ERNIE 语义的混合结构模块组合实验

根据以上发现,为了探究基于 ERNIE 语义的高性能混合结构设计,本文接下来进行了表4.19的混合结构模块组合实验。

表 4.19 基于 ERNIE 嵌入的混合结构模块组合实验结果

模块名称	选择情况									
	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Dense	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Flatten	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CAM(Hidden)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SFC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
FC-Drop ^a	✓	✓		✓	✓	✓	✓	✓	✓	✓
FC										
BiLSTM(B)	✓	✓		✓	✓	✓	✓	✓	✓	✓
LayerNorm				✓	✓	✓	✓	✓	✓	✓
SAM	✓				✓	✓	✓	✓	✓	✓
MSA		✓		✓						
BiLSTM(A)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Conv-Pool2 ^b				✓		✓	✓	✓	✓	✓
CAM									✓	✓
Conv-Pool1 ^c							✓		✓	
ResLn-Pool ^d										✓
准确率	0.7444 ±0.0615	0.7333 ±0.0636	0.7400 ±0.0938	0.7400 ±0.0717	0.7622 ±0.0731	0.7489 ±0.0696	0.7156 ±0.0842	0.7111 ±0.1052	0.7489 ±0.0757	0.7311 ±0.0655
精确率	0.7394 ±0.0724	0.7360 ±0.0867	0.7358 ±0.0964	0.7283 ±0.0799	0.7652 ±0.0781	0.7680 ±0.0807	0.7259 ±0.0975	0.7290 ±0.1172	0.7582 ±0.0824	0.7512 ±0.0818
召回率	0.7674 ±0.1089	0.7587 ±0.1076	0.7540 ±0.1382	0.7769 ±0.0957	0.7595 ±0.1011	0.7202 ±0.1208	0.7190 ±0.1226	0.7142 ±0.1502	0.7419 ±0.1072	0.7144 ±0.1086
F1 分数	0.7479 ±0.0667	0.7388 ±0.0555	0.7396 ±0.1020	0.7482 ±0.0704	0.7600 ±0.0795	0.7377 ±0.0842	0.7142 ±0.0908	0.7092 ±0.1065	0.7454 ±0.0780	0.7248 ±0.0639
AUC	0.7431 ±0.0618	0.7324 ±0.0629	0.7395 ±0.0937	0.7397 ±0.0712	0.7618 ±0.0731	0.7493 ±0.0695	0.7149 ±0.0838	0.7106 ±0.1039	0.7490 ±0.0752	0.7308 ±0.0642

^a 丢弃隐藏单元概率 $\theta = 0.3$

^b 滤波器个数 $n = 128$

^c 滤波器个数 $n = 64$

^d 滤波器个数 $F = 64$

基于表4.19所示的 ERNIE 混合结构模块组合实验结果表明,当集成 BiLSTM(A/B)、LayerNorm 和 SAM 模块时(第 5 列配置),模型在准确率($76.22\% \pm 7.31\%$)、精确率($76.52\% \pm 7.81\%$)和 F1 分数($76.00\% \pm 7.95\%$)三个核心指标上均达到峰值,其中准确率较基准 BiLSTM(A/B)、SAM 模块配置(第 1 列)显著提升 1.78 个百分点。值得注意的是,该配置通过 LayerNorm 与 SAM 的协同优化,使召回率标准差(10.11%)较未使用 LayerNorm 的同类配置(第 2 列)降低 2.65 个百分点,展现出知识增强特征与归一化技术的特殊适配性。

基于 ERNIE 嵌入的混合结构模块组合实验结果分析表明，ERNIE 混合框架的性能提升主要源于两个核心机制：其一是 LayerNorm 层与自注意力模块（SAM）在时空维度上的高效协作，其二是双向 LSTM 结构通过双通道机制实现的语义互补效应。值得注意的是，在 ERNIE 的语义表征框架下，传统逻辑回归（LR）模型仍展现出优异的分类性能。

基于此发现，本研究创新性地将第五列网络层配置方案与 LR 算法进行融合，最终构建出具有层次化特征的 DepHybrid 复合模型架构。这种设计策略不仅继承了 ERNIE 的深度表征优势，同时充分发挥了经典机器学习模型的高效推理特性。

5. DepHybrid 与现有架构的性能对比实验

为验证 DepHybrid 架构的性能优势，本文在相同实验条件下复现了多种现有方法进行对比分析，结果如表4.20所示。各对比模型的技术特性可归纳如下：

TextCNN^[96] 作为经典文本分类框架，通过多尺度卷积核提取局部 n-gram 特征。CapsNet^[113] 创新性地采用胶囊结构与动态路由算法，突破传统 CNN 在空间关系建模的局限。A-Hybrid^[114] 构建了包含 BiLSTM、LSTM、卷积层和 Loong 注意力层的混合架构，CBA^[115] 则通过在 CNN-BiLSTM 基础上集成 Loong 注意力机制提升性能。

BertGCN^[116] 将 BERT 的语义表征与 GCN 的图结构学习相结合，M-BERT^[117] 通过 MAG 机制融合 BERT 与 LDA 特征。HBGA^[118] 采用 BERT 嵌入后，通过 BiLSTM、胶囊网络和 GAT 分别捕获全局依赖、局部语义与类别关联。AMCDD^[119] 则基于 BERT 嵌入构建 LSTM-BiGRU 双通道结构进行抑郁识别。

相较于上述方法，DepHybrid 通过并行 LR 与 BiLSTM 分支，同步捕捉局部序列模式与全局统计特征；引入可学习的通道注意力机制，自主校准多源特征通道的权重分布；采用加权概率融合策略平衡深层特征表达能力与模型泛化性能。

表 4.20 基于 ERNIE 嵌入的 DepHybrid 与现有架构性能表现

发表年份	架构名称	准确率	精确率	召回率	F1 分数	AUC
2014	TextCNN	0.7356±0.0816	0.7166±0.0886	0.7905±0.0685	0.7504±0.0737	0.7355±0.0820
2017	CapsNet	0.6956±0.0911	0.6382±0.2302	0.7144±0.2567	0.6645±0.2249	0.7731±0.0980
2021	BertGCN	0.6978±0.0499	0.7215±0.0789	0.6617±0.0566	0.6867±0.0460	0.6974±0.0507
2022	HBGA	0.6933±0.0988	0.7359±0.1217	0.6038±0.1622	0.6546±0.1281	0.6934±0.1000
2023	A-Hybrid	0.7356±0.0792	0.7519±0.1159	0.7490±0.1235	0.7383±0.0727	0.7343±0.0791
2023	M-BERT	0.5956±0.1291	0.3316±0.3201	0.6180±0.3513	0.2955±0.3150	0.5974±0.1273
2024	CBA	0.5311±0.0557	0.3213±0.2651	0.5111±0.4239	0.3927±0.3228	0.5318±0.0511
2025	AMCDD	0.7289±0.0824	0.7308±0.1033	0.7551±0.0898	0.7367±0.0715	0.7290±0.0818
Proposed	DepHybrid	0.7689±0.0790	0.7733±0.0876	0.7632±0.1062	0.7658±0.0862	0.7681±0.0789

如表4.20所示，DepHybrid 与现有架构的性能对比实验验证了 DepHybrid 架构相对于现有主流文本分类架构的性能优势。在 ERNIE 预训练嵌入的基础上，DepHybrid 架构在准确率（76.89%±0.079）、精确率（77.33%±0.0876）、F1 分数（76.58%±0.0862）和 AUC 值

($76.81\% \pm 0.0789$) 四项核心指标上均取得最优表现，展现出均衡且稳定的分类性能。实验结果充分证明，DepHybrid 通过深度融合依存句法结构信息与混合网络架构，有效提升了文本表征的判别能力，在保持各评价指标均衡发展的同时实现了分类性能的全面提升。

通过基于 BoW、BERT、RoBERTa 和 ERNIE 特征的模型设计选择实验可以得出，基于 ERNIE 的 DepHybrid 框架可以充分适配 ERNIE 语义编码，实现了较现有架构分类性能的有效提升。

4.5.5 访谈问答数据选用对比实验

为了评估不同访谈问题在抑郁症判别中的特征表征能力与分类贡献度，从而量化解析各语义单元对心理健康评估的预测效力，本文接下来进行数据选用方面的讨论。访谈问题序号与内容映射情况见表4.21，图4.25与图4.26反映了各访谈问题的抑郁识别情况。

表 4.21 访谈问题序号与内容映射表^[120,121]

序号	内容
1	如果您有一个假期可以去旅游，请描述一下您的旅游计划。
2	请跟我们分享一段您认为美好的回忆。大致描述一下当时的场景。
3	您最近情绪如何？这种情绪对您的生活带来了什么影响？
4	您最近身体状况如何？对您的生活有哪些影响？
5	您如何评价自己？
6	请讲一讲让您感觉很糟糕的经历。请描述一件让您非常痛苦的事情。
7	正性表情图片描述
8	中性表情图片描述
9	负性表情图片描述

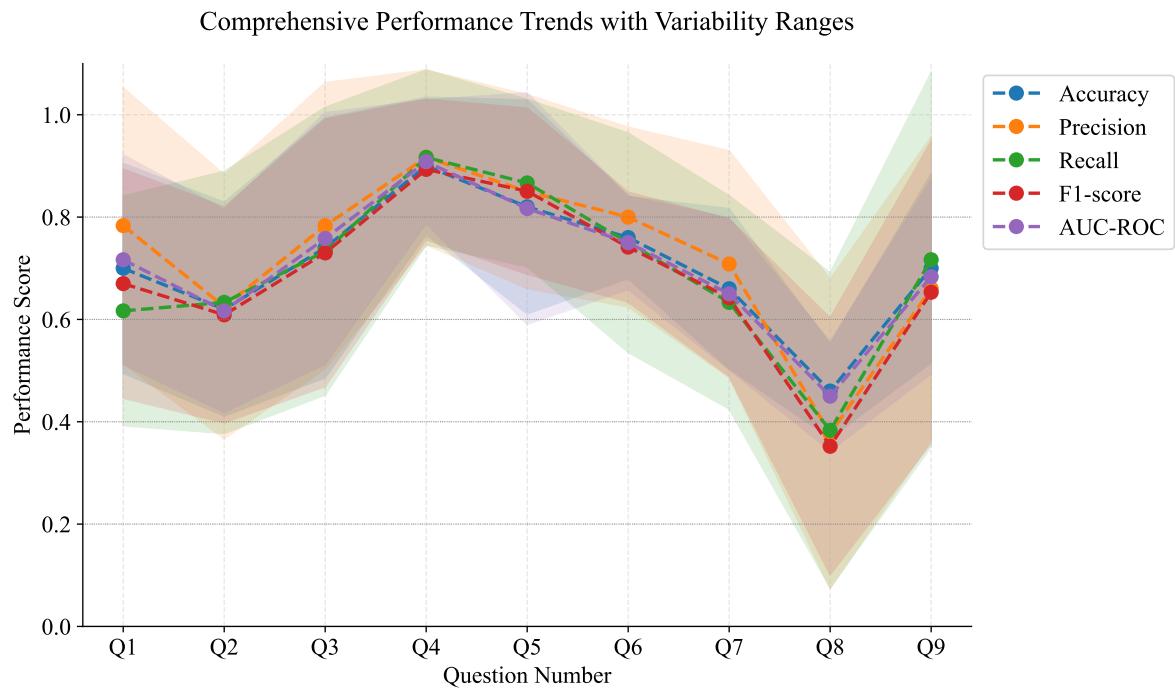


图 4.25 各访谈问题的综合性能趋势与变异范围

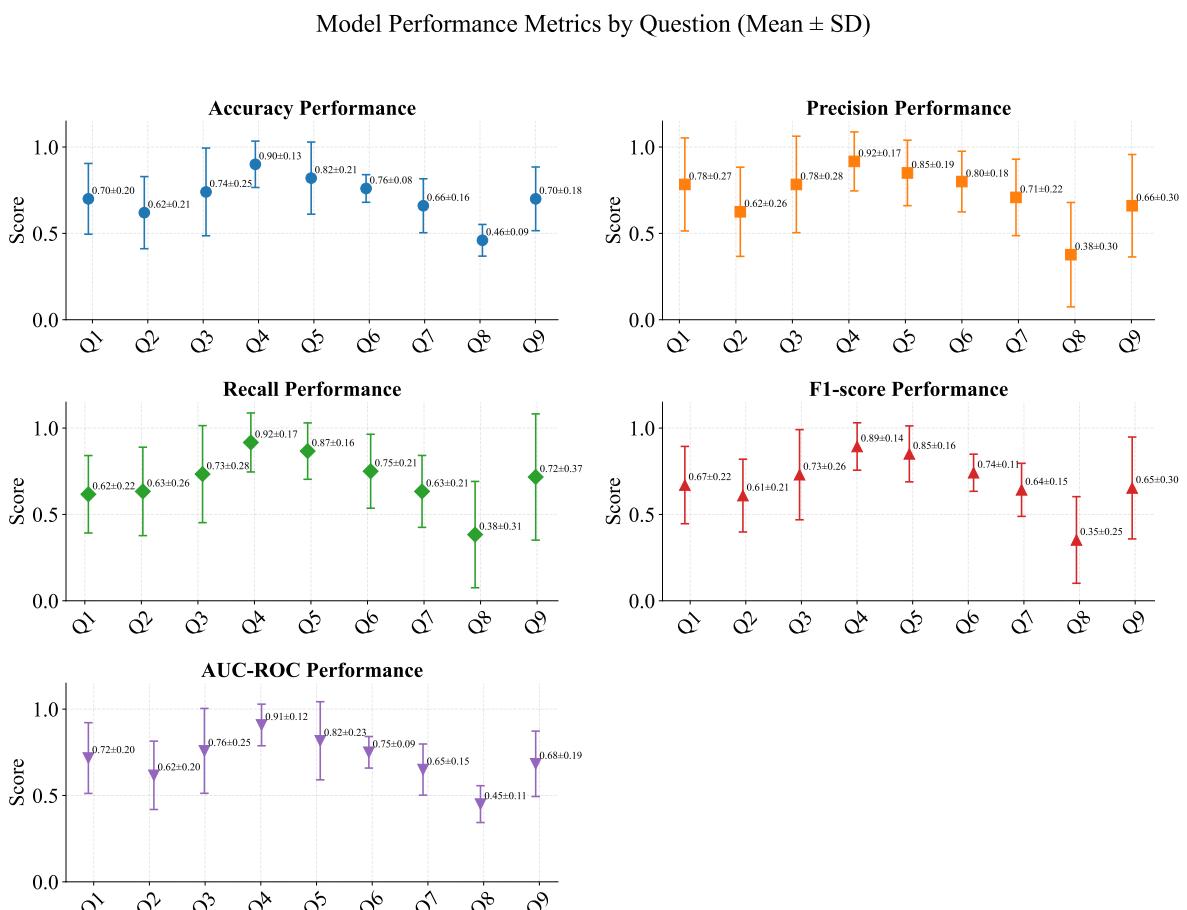


图 4.26 各访谈问题对于抑郁识别性能的贡献情况分析

通过系统分析访谈问题设计对心理状态分类模型性能的影响，本文发现不同提问维度存在显著的效能差异。从模型表现来看，问题4（身体状况描述）展现最佳分类效果，其准确率达0.9000 (± 0.1342)，F1分数0.8933 (± 0.1373)，AUC-ROC值0.9083 (± 0.1205)，显著优于其他问题。这或源于生理症状描述具有较高的客观性与表达一致性，如“头痛”、“失眠”等典型症状的词汇特征易于模型捕捉。

相较而言，中性表情图片描述（问题8）的分类效能最低，准确率仅为0.4600 (± 0.0917)，与随机猜测水平相当。这可能与中性表情本身的情感模糊性相关，被试在描述时易产生歧义表达，导致模型难以建立有效的特征关联。值得注意的是，负性情绪相关问题的表现呈现分化态势：创伤经历回忆（问题6）取得中等准确率0.7600 (± 0.0800)，而负性表情描述（问题9）的F1分数下降至0.6531 (± 0.2951)，暗示情绪体验的主观表述与表情识别的客观描述之间存在表征差异。

在自我认知维度，自我评价问题（问题5）展现出了较高的预测效度（AUC-ROC为 0.8167 ± 0.2261 ），可能受益于被试在自我描述中无意识地使用具有心理诊断价值的语言模式。而正性情绪相关问题的分类表现呈现梯度差异：旅游计划描述（问题1）准确率0.7000 (± 0.2049) 显著高于正性表情描述（问题7）的0.6600 (± 0.1562)，提示具体情境叙述比抽象表情解读更有利于模型进行特征提取。

标准差分析显示，回忆类问题（问题2、6）的指标离散度较高（F1分数SD分别达0.2108和0.1075），反映个体在创伤经历与美好回忆的表述方式上存在较大异质性。

通过访谈问答数据选用对比实验可以归纳出，问题设计的指向性强度与应答内容的客观化程度共同构成影响分类效能的关键因素，其中生理症状描述等客观化问题具有最优特征表征稳定性，情况问诊的效果整体上要优于图片描述，负性和中性问诊问题的效果优于正性问题。

4.5.6 案例分析实验

本研究的案例分析遵循“数据-模型-决策”的验证路径：首先通过特征分布分析确保数据表征合理性（数据层）；继而从泛化能力和参数优化维度验证模型架构的有效性（模型层）；最后通过文本解释性研究打通模型输出与领域知识的映射（决策层）。这一递进式设计不仅满足方法论的完备性要求，更从技术可复现性、结果可追溯性层面回应了可信人工智能（Trustworthy AI）的研究范式。

1. 基于t-SNE的特征分布可视化

本研究通过系统性实验证明了DepHybrid架构的识别优势。为进一步揭示其内在决策机理，采用特征空间可视化方法对架构的输出表征进行深度解析。由于二维投影将抽象特征转化为人类可直观解析的空间模式（如聚类、离群点），为验证特征判别性与决策逻辑提供视觉证据，因此采用t-SNE^[122]降维技术将其映射到二维平面，以便进行直观的可视化呈现。t-SNE相较于PCA等线性方法，更适用于深度学习模型中高度非线性变换后的特征

表示分析。图4.27展示了数据集训练前后的 ERNIE 特征可视化分布图，可以看出有一定的类别可分性，说明数据集中含有噪声，不同访谈问题的性能贡献程度是不同的。

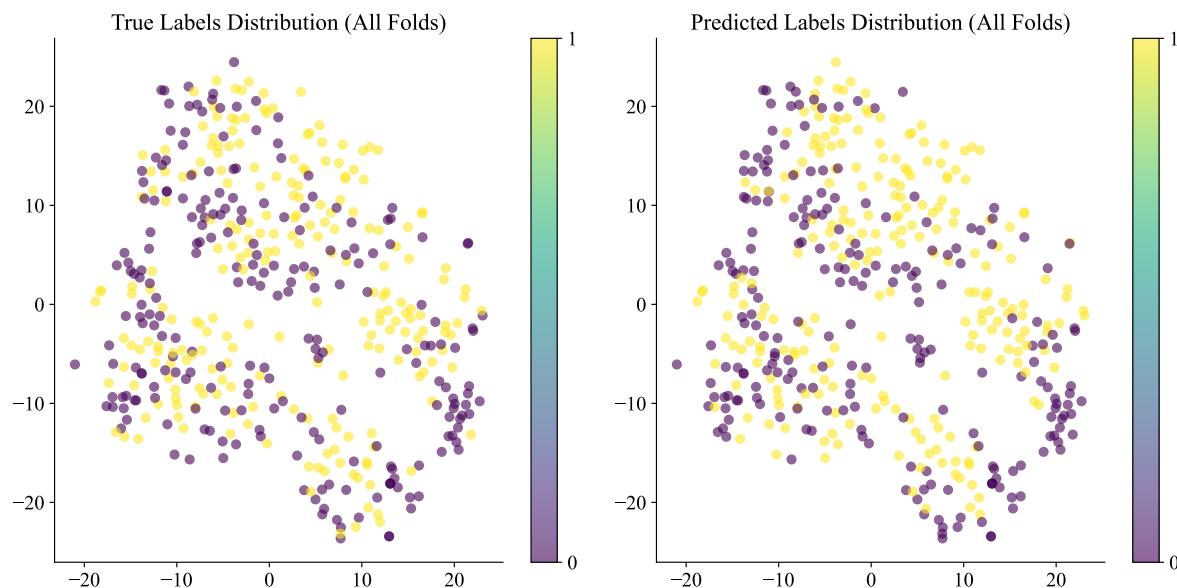


图 4.27 数据集训练前后的 ERNIE 特征可视化分布图

图4.27的特征分布可视化分布图展示了交叉训练中每一折预测值与真实值对应的特征降维后的可视化分布。从中可以看出左右两边呈现高度一致的分布，说明模型的预测效果较好，同时也验证了 ERNIE 的特征表示是可被分类识别的。

2. 基于特定访谈问题的泛化水平对比实验

由4.5.5节可知，问题 4 对于抑郁识别的性能贡献度最高，为了分析访谈问题间的情感关联，本节选用问题 4 讨论其泛化水平。

表 4.22 访谈问题 4 泛化水平的性能表现

访谈问题序号	准确率	精确率	召回率	F1 分数	AUC
1	0.6200±0.2272	0.6050±0.2263	1.0000±0.0000	0.7310±0.1635	0.6417±0.1974
2	0.5600±0.1744	0.5450±0.1540	1.0000±0.0000	0.6929±0.1270	0.5750±0.1146
3	0.8200±0.1077	0.8250±0.1843	0.9000±0.1528	0.8324±0.0952	0.8417±0.0946
4	0.9000±0.1342	0.9167±0.1708	0.9167±0.1708	0.8933±0.1373	0.9083±0.1205
5	0.6000±0.1549	0.5650±0.1519	0.9167±0.1708	0.6864±0.1418	0.6083±0.1181
6	0.5400±0.1281	0.5317±0.1226	0.9667±0.1000	0.6726±0.0965	0.5500±0.0850
7	0.6200±0.2272	0.6217±0.2241	0.9667±0.1000	0.7310±0.1635	0.6333±0.1979
8	0.5400±0.1562	0.5567±0.1719	0.9000±0.1528	0.6624±0.0973	0.5500±0.1675
9	0.5200±0.1833	0.5150±0.1534	0.9167±0.1708	0.6500±0.1520	0.5250±0.1539

根据表4.21和表4.22的实验结果对比分析，可以发现基于访谈问题 4（“您最近身体状

况如何？对您的生活有哪些影响？”）建立的分类模型展现出独特的泛化特性。首先，问题4作为训练集时，其自身验证指标显著优于其他问题测试结果，准确率达 0.9000 ± 0.1342 ，F1分数达 0.8933 ± 0.1373 ，AUC值达 0.9083 ± 0.1205 ，表明该问题具有较好的内部一致性特征。值得注意的是，模型在问题3（情绪状态）和问题7（正性表情描述）的测试中分别取得次优的准确率（ 0.8200 ± 0.1077 和 0.6200 ± 0.2272 ），提示生理状态与情绪表达之间可能存在潜在特征关联。

实验数据同时揭示模型的泛化能力存在领域特异性：在涉及具体生活事件描述的问题（1、2、5、6）中，召回率普遍高于精确率（如问题6的召回率达 0.9667 ± 0.1000 但精确率仅 0.5317 ± 0.1226 ），反映模型对负面生活事件的敏感性；而对于中性和负性表情图片描述（问题8、9），各项指标均呈现最低值（AUC分别为 0.5500 ± 0.1675 和 0.5250 ± 0.1539 ），说明非语言信息的跨模态识别存在显著挑战。这种性能差异可能源于问题4的生理状态描述具有更明确的语言特征标记，而视觉刺激的文本描述则涉及更复杂的多模态特征映射。

通过基于特定访谈问题的泛化水平对比实验可以得出，模型对负面生活事件具有过度敏感特性，生理状态与情绪表达之间可能存在潜在特征关联，情况问诊的效果整体上要优于图片描述。

3. 基于LR模型的UCB参数优化方法与其他方法的对比实验

为了验证UCB算法对参数调优的有效性，而liblinear^[123]同时支持L1, L2正则化，因此本文以基于liblinear求解器的LR参数优化为例，进行了表4.23的对比实验。

表4.23 对ERNIE-LR模型进行参数优化的性能表现

优化方法	准确率	精确率	召回率	F1分数	AUC
无优化	0.7533 ± 0.0740	0.7484±0.0750	0.7630 ± 0.1226	0.7519 ± 0.0874	0.7526 ± 0.0745
Grid Search	0.6889 ± 0.1004	0.6791 ± 0.1012	0.8049 ± 0.1190	0.7244 ± 0.0607	0.7966 ± 0.0563
Random Search	0.6489 ± 0.0993	0.6366 ± 0.1047	0.8445 ± 0.1412	0.7092 ± 0.0514	0.7654 ± 0.0600
UCB	0.7556 ± 0.1052	0.7422 ± 0.1233	0.8666 ± 0.0888	0.7869 ± 0.0594	0.7970 ± 0.0874

根据表4.23中ERNIE-LR模型参数优化的对比实验结果，可以得到以下结论：基于UCB算法的参数优化方法在多项性能指标上展现出显著优势。具体而言，UCB在准确率（0.7556）、召回率（0.8666）、F1分数（0.7869）和AUC（0.7970）四个核心指标上均取得最高值，其中F1分数较传统网格搜索和随机搜索分别提升8.6%和10.9%，表明该方法能更有效地平衡精确率与召回率。尽管无优化方法在精确率（0.7484）指标上略优于UCB（0.7422），但其召回率（0.7630）显著低于UCB（0.8666），显示出UCB在保持较高预测精度的同时显著提升了模型对正样本的识别能力。与网格搜索和随机搜索相比，UCB的AUC值分别提高0.4%和4.1%，且各指标的标准差（ $\pm0.0594\pm0.1052$ ）处于合理范围，说明该方法具有较好的鲁棒性。

实验结果表明，UCB算法通过动态平衡探索与利用，能够更有效地实现参数空间的智

能搜索，为模型性能优化提供了一种高效解决方案。

4. 基于 SHAP 的文本解释可视化

针对传统特征重要性分析在高维文本数据中的局限性，本研究采用基于 Shapley 值的 SHAP 解释方法。该方法通过计算每个词汇特征对预测结果的边际贡献，特别适用于分析文本特征间的非线性交互作用^[124]。图4.28至图4.32展示了数据集前五个样本的词汇贡献可视化结果，其中颜色深浅表征影响强度：红色表示积极影响（SHAP 值为正），蓝色表示消极影响（SHAP 值为负）。这种可解释性分析方法为临床实践提供了双重价值：既验证了模型决策的合理性，又辅助精神科医生在文本中定位具有诊断价值的语言特征。

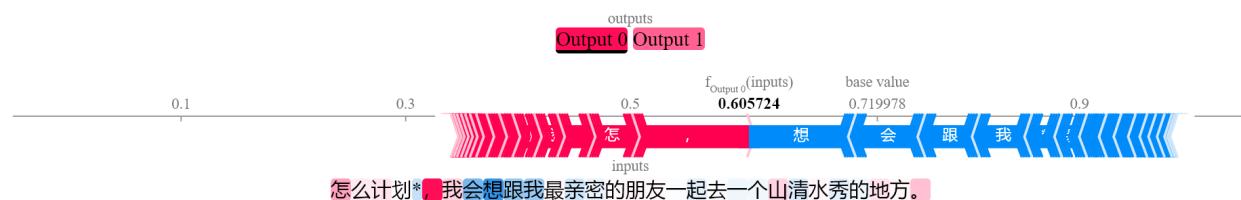


图 4.28 第一个样本的文本解释图

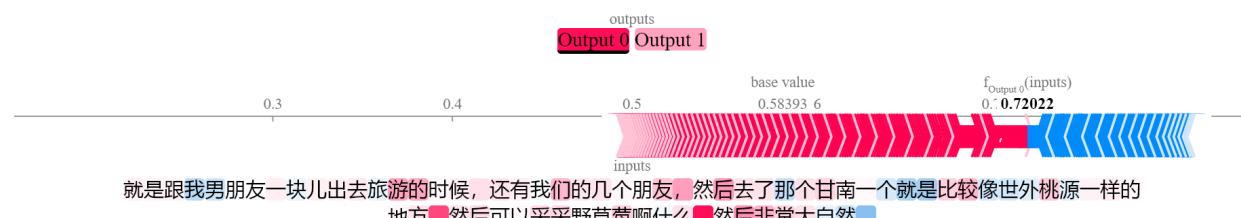


图 4.29 第二个样本的文本解释图



图 4.30 第三个样本的文本解释图

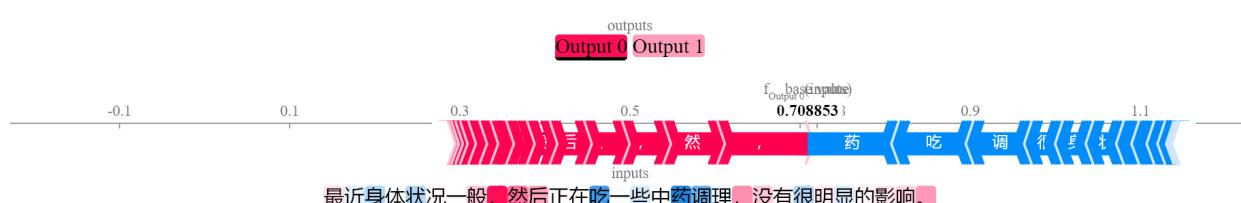


图 4.31 第四个样本的文本解释图



图 4.32 第五个样本的文本解释图

基于 SHAP 的文本解释与可视化分析表明, DepHybrid 架构能够有效识别文本中的消极词汇特征, 这些关键特征的捕捉显著提升了模型在抑郁识别任务中的分类性能。

以上案例分析实验验证了本文模型架构的可信度, 为临床决策支持与心理健康服务普惠化提供了新路径。

4.6 实验结论

本章提出了一种基于抑郁症识别的关键影响因素分析框架, 经过上述的实验, 可以得到以下结论:

- (1) LR 可作为抑郁识别分类任务的基线模型, 大部分模型在不同的文本截断参数设置下均呈现稳定的性能表现, 因此文本截断参数可视为非敏感性超参数。
- (2) 优秀的特征生成器需满足基础性能达标与跨模块稳定性突出双重条件, BoW、BERT、RoBERTa 和 ERNIE 四类模型可有效构造文本特征。
- (3) 在基于 FNN 框架的模块化组合测试中, CAM 与 SFC 模块展现出显著的互补效应, 同时采用复杂度适中的卷积架构被证实能够有效增强模型的文本分类精度; 而在 RNN 架构中, BiLSTM 与 SFC 在时序特征提取方面形成互补优势, 而 CAM 模块的层级特征强化作用则为分类任务提供了额外的性能增益。
- (4) ERNIE 语义编码方法在与 FNN 及 RNN 结构的组合实验中, 其准确率、召回率等核心评测维度均呈现显著优势, LayerNorm 层与 SAM 模块的结合与双层 BiLSTM 协同效果较好, 实验进一步提出的 DepHybrid 架构可很好地适配 ERNIE 特征, 有效提升抑郁识别性能。
- (5) 访谈文本中关于身体状况的描述可有效识别抑郁状况, 图片描述型文本则不能较好地反映抑郁趋势, 性能较差, 负性和中性问诊问题的效果优于正性问题。
- (6) ERNIE 的特征表示可被较好地分类识别; UCB 算法可有效优化模型参数, 提高模型的稳定性并保持较好的性能; 身体状况与情绪状态语义关联度较高, 推理能力最强; DepHybrid 架构能够有效识别文本中的消极词汇特征。

4.7 本章小结

本章节主要聚焦于基于深度学习的抑郁情感分析任务，采用目前常用的词嵌入模型和深度神经网络结构，设计控制变量实验以量化评估特征构造策略、模型架构选择、任务范式优化及泛化能力水平对算法性能的贡献情况。本章节详细介绍了控制变量实验的整个过程以及变量使用方法。在实验部分，本章节通过与多个基线变量对比，验证论文架构的有效性，展示其在抑郁情感分析任务上的优越性能。

第五章 总结与展望

5.1 总结

本研究围绕中文抑郁症识别任务中的关键影响因素展开系统性探索，针对现有研究在数据资源、模型解释性与优化方法上的不足，从数据构建、模型架构设计到参数优化三个维度进行了创新性研究，主要完成了以下工作：

- (1) 本文使用了兰州大学合作医院采集的标准化临床访谈录音，经自动语音识别与专业人工标注，构建了一个经临床验证的中文抑郁访谈文本数据集 DepInter-CN。
- (2) 本文从特征工程、模型架构、数据选用三个方面构建影响因素分析框架，在此之前本文通过分别选用离散式表示与分布式表示的代表性方法构建文本特征空间并使用一系列机器学习模型进行预实验，筛选出可用于后续实验的基线变量。同时考虑到因素间存在的相互影响，通过排除法逐步排除较差表现的自变量。
- (3) 本文通过词袋模型与预训练语言模型的识别性能对比实验，揭示了中文语义与文本特征空间的适配情况，发现了 ERNIE 模型良好的表征学习能力。
- (4) 本文设计了一系列深度学习模块组合实验，评估各模型与特征的融合效果，最终提出了一种混合架构 DepHybrid，达到了 77% 的准确率及 F1 分数。
- (5) 本文通过解构访谈问题的语义组合，分析各类问题得到的情感线索，发现情况问诊的效果整体上要优于图片描述，负性和中性问诊问题的效果优于正性问题。
- (6) 本文引入了多臂老虎机理论，设计基于 UCB 的超参数自适应优化器，使准确率提高了 0.23%，召回率提升了近 10.3%，AUC 提升了近 4.5%。
- (7) 本文使用了 t-SNE 与 SHAP 方法进行可视化特征分布及特征重要程度，验证了模型架构的可信度。

5.2 未来展望

本研究在中文抑郁症识别任务的多维度优化与可解释性分析方面取得了一定进展，但仍需在以下方向进行深入探索：

- (1) 数据构建方面，未来可扩大 DepInter-CN 数据集的覆盖范围，增加不同地区方言、年龄层和抑郁亚型的临床案例，同时探索将语音特征与文本信息相结合的多模态分析方法，更完整地呈现抑郁症的语言特征。
- (2) 模型设计上，建议开发更适配中文特点的预训练模型优化方法，引入对比学习等新方法提升模型识别情感特征的能力。针对临床访谈场景，可研究基于强化学习的智能提问优化模型，动态调整问题路径。
- (3) 现有参数优化方法可拓展到模型结构设计领域，尝试将神经架构搜索（NAS）与

超参数优化相结合的统一框架。

(4) 临床应用方面，需要建立模型预测结果与临床诊断标准之间的对应关系，开发实用的辅助决策系统，并通过跨医院临床试验验证实际效果。

(5) 在伦理规范方面，需要深入研究人机协作诊断中的责任归属问题，完善符合医疗伦理的数据匿名化处理与隐私保护方案。这些改进将提升抑郁症识别技术的准确性和可靠性，为智能精神健康诊断提供更有力的技术支持。

参考文献

- [1] 医学名词审定委员会精神医学名词审定分委员会. 精神医学名词 [M]. 北京: 科学出版社, 2019.
- [2] Beck A T, Alford B A. *Depression: Causes and Treatment*[M]. University of Pennsylvania Press, 2009.
- [3] 世界卫生组织. 抑郁障碍 (抑郁症) [EB/OL]. <https://www.who.int/zh/news-room/fact-sheets/detail/depression>.
- [4] Paykel E S. Basic Concepts of Depression[J]. *Dialogues in Clinical Neuroscience*, 2008, 10(3):279–289.
- [5] Kalachanis K, Michailidis I E. The Hippocratic View on Humors and Human Temperament[J]. *European Journal of Social Behaviour*, 2015, 2(2):1–5.
- [6] 田代华. 黄帝内经素问 [M]. 人民卫生出版社, 2007.
- [7] Johnstone M. *I Had a Black Dog: His Name was Depression*[M]. Pan, 2005.
- [8] Vos T, Lim S S, Abbafati C, et al. Global Burden of 369 Diseases and Injuries in 204 Countries and Territories, 1990–2019: A Systematic Analysis for the Global Burden of Disease Study 2019[J]. *The Lancet*, 2020, 396(10258):1204–1222.
- [9] Huang Y, Wang Y, Wang H, et al. Prevalence of Mental Disorders in China: A Cross-sectional Epidemiological Study[J]. *The Lancet Psychiatry*, 2019, 6(3):211–224.
- [10] Lu J, Xu X, Huang Y, et al. Prevalence of Depressive Disorders and Treatment in China: A Cross-sectional Epidemiological Study[J]. *The Lancet Psychiatry*, 2021, 8(11):981–990.
- [11] 马晓梅, 王瑾瑾, 徐学琴, 等. 中国居民 1990 年与 2019 年抑郁症疾病负担情况比较 [J]. 中国公共卫生, 2022, 38(10):1345–1347.
- [12] Qin X, Pan J. The Medical Cost Attributable to Obesity and Overweight in China: Estimation based on Longitudinal Surveys[J]. *Health Economics*, 2016, 25(10):1291–1311.
- [13] Hsieh C R, Qin X. Depression Hurts, Depression Costs: The Medical Spending Attributable to Depression and Depressive Symptoms in China[J]. *Health Economics*, 2018, 27(3):525–544.
- [14] Olesen J, Gustavsson A, Svensson M, et al. The Economic Cost of Brain Disorders in Europe[J]. *European Journal of Neurology*, 2012, 19(1):155–162.
- [15] Greenberg P E, Fournier A A, Sisitsky T, et al. The Economic Burden of Adults with Major Depressive Disorder in the United States (2010 and 2018)[J]. *Pharmacoconomics*, 2021, 39(6):653–665.
- [16] 中华人民共和国国家卫生健康委员会统计信息中心. 中国卫生健康统计年鉴 [R/OL]. 中华人民共和国国家卫生健康委员会, 2021. <http://www.nhc.gov.cn/mohwsbwstjxxzx/tjtjn/202305/304a301bfd/b444afbf94b1a6c7f83bca.shtml>.
- [17] APA. *Diagnostic and Statistical Manual of Mental Disorders*[M]. American Psychiatric Publishing, 2000.
- [18] 世界卫生组织. *ICD-11*[EB/OL]. <https://icd.who.int/zh>.
- [19] Kroenke K, Spitzer R L, Williams J B. The PHQ-9: Validity of a Brief Depression Severity Measure[J]. *Journal of General Internal Medicine*, 2001, 16(9):606–613.
- [20] Maj M, Stein D J, Parker G, et al. The Clinical Characterization of the Adult Patient with Depression Aimed at Personalization of Management[J]. *World Psychiatry*, 2020, 19(3):269–293.

- [21] Mundt J C, Snyder P J, Cannizzaro M S, et al. Voice Acoustic Measures of Depression Severity and Treatment Response Collected via Interactive Voice Response (IVR) Technology[J]. *Journal of Neurolinguistics*, 2007, 20(1):50–64.
- [22] Nock M K, Borges G, Bromet E J, et al. Suicide and Suicidal Behavior[J]. *Epidemiologic Reviews*, 2008, 30(1):133.
- [23] Luscher B, Shen Q, Sahir N. The GABAergic Deficit Hypothesis of Major Depressive Disorder[J]. *Molecular Psychiatry*, 2011, 16(4):383–406.
- [24] He L, Niu M, Tiwari P, et al. Deep Learning for Depression Recognition with Audiovisual Cues: A Review[J]. *Information Fusion*, 2022, 80:56–86.
- [25] Ma X, Yang H, Chen Q, et al. DepAudioNet: An Efficient Deep Model for Audio based Depression Classification[C]. *Proceedings of the 6th International Workshop on Audio/Visual Emotion Challenge*. 2016. 35–42.
- [26] Al Jazaery M, Guo G. Video-based Depression Level Analysis by Encoding Deep Spatiotemporal Features[J]. *IEEE Transactions on Affective Computing*, 2018, 12(1):262–268.
- [27] Park M, Cha C, Cha M. Depressive Moods of Users Portrayed in Twitter[C]. *Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data Mining, SIGKDD 2012*. 2012. 1–8.
- [28] De Choudhury M, Gamon M, Counts S, et al. Predicting Depression via Social Media[C]. *Proceedings of the International AAAI Conference on Web and Social Media*. 2013. 128–137.
- [29] Yang Q, Wang Z, Chen H, et al. PsychoGAT: A Novel Psychological Measurement Paradigm through Interactive Fiction Games with LLM Agents[C]. *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2024. 14470–14505.
- [30] APA. *Diagnostic and Statistical Manual of Mental Disorders: DSM-5*[M], volume 5. American Psychiatric Association Washington, DC, 2013.
- [31] Cadoret R J, Troughton E, Merchant L M, et al. Early Life Psychosocial Events and Adult Affective Symptoms[M]. *Straight and Devious Pathways from Childhood to Adulthood*. New York, NY, US: Cambridge University Press, 1990. 300–313.
- [32] Russell J A. A Circumplex Model of Affect.[J]. *Journal of Personality and Social Psychology*, 1980, 39(6):1161.
- [33] Yu L C, Lee L H, Hao S, et al. Building Chinese Affective Resources in Valence-arousal Dimensions[C]. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2016. 540–545.
- [34] Likert R. A Technique for the Measurement of Attitudes[J]. *Archives of Psychology*, 1932.
- [35] Hamilton M. A Rating Scale for Depression[J]. *Journal of Neurology, Neurosurgery, and Psychiatry*, 1960, 23(1):56.
- [36] Beck A T, Ward C H, Mendelson M, et al. An Inventory for Measuring Depression[J]. *Archives of General Psychiatry*, 1961, 4(6):561–571.
- [37] Zung W W. A Self-rating Depression Scale[J]. *Archives of General Psychiatry*, 1965, 12(1):63–70.
- [38] Montgomery S A, Åsberg M. A New Depression Scale Designed to be Sensitive to Change[J]. *The British Journal of Psychiatry*, 1979, 134(4):382–389.

- [39] Rush A J, Gullion C M, Basco M R, et al. The Inventory of Depressive Symptomatology (IDS): Psychometric Properties[J]. *Psychological Medicine*, 1996, 26(3):477–486.
- [40] Rush A J, Trivedi M H, Ibrahim H M, et al. The 16-Item Quick Inventory of Depressive Symptomatology (QIDS), Clinician Rating (QIDS-C), and Self-report (QIDS-SR): A Psychometric Evaluation in Patients with Chronic Major Depression[J]. *Biological Psychiatry*, 2003, 54(5):573–583.
- [41] Ay B, Yildirim O, Talo M, et al. Automated Depression Detection using Deep Representation and Sequence Learning with EEG Signals[J]. *Journal of Medical Systems*, 2019, 43:1–12.
- [42] Alghowinem S, Goecke R, Wagner M, et al. Multimodal Depression Detection: Fusion Analysis of Paralinguistic, Head Pose and Eye Gaze Behaviors[J]. *IEEE Transactions on Affective Computing*, 2016, 9(4):478–490.
- [43] Sardari S, Nakisa B, Rastgoo M N, et al. Audio Based Depression Detection using Convolutional Autoencoder[J]. *Expert Systems with Applications*, 2022, 189:116076.
- [44] 李世琪, 刁宇峰, 张浩, 等. 基于层次数据增强的多维度特征融合社交媒体抑郁症识别 [J/OL]. 计算机工程与应用, 1–12. <http://kns.cnki.net/kcms/detail/11.2127.TP.20241111.0946.013.html>.
- [45] Yoon J, Kang C, Kim S, et al. D-vlog: Multimodal Vlog Dataset for Depression Detection[C]. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2022. 12226–12234.
- [46] Ansari L, Ji S, Chen Q, et al. Ensemble Hybrid Learning Methods for Automated Depression Detection[J]. *IEEE Transactions on Computational Social Systems*, 2022, 10(1):211–219.
- [47] Cohen A S, Rodriguez Z, Warren K K, et al. Natural Language Processing and Psychosis: On the Need for Comprehensive Psychometric Evaluation[J]. *Schizophrenia Bulletin*, 2022, 48(5):939–948.
- [48] Mitchell M, Hollingshead K, Coppersmith G. Quantifying the Language of Schizophrenia in Social Media[C]. *Proceedings of the 2nd Workshop on Computational Linguistics and Clinical Psychology: From Linguistic Signal to Clinical Reality*. 2015. 11–20.
- [49] Irving J, Patel R, Oliver D, et al. Using Natural Language Processing on Electronic Health Records to Enhance Detection and Prediction of Psychosis Risk[J]. *Schizophrenia Bulletin*, 2021, 47(2):405–414.
- [50] Freud S. Psychopathology of Everyday Life (AA Brill, Trans.) New York: New American Library[J]. *Original work published*, 1901.
- [51] Rorschach H. Psychodiagnostik[J]. *The Journal of Nervous and Mental Disease*, 1922, 56(3):306.
- [52] McClelland D C. Inhibited Power Motivation and High Blood Pressure in Men[J]. *Journal of Abnormal Psychology*, 1979, 88(2):182.
- [53] Winter D G. A Motivational Analysis of the Clinton First Term and the 1996 Presidential Campaign[J]. *The Leadership Quarterly*, 1998, 9(3):367–376.
- [54] Holtzman W H. Validation Studies of the Rorschach Test: Shyness and Gregariousness in the Normal Superior Adult[J]. *Journal of Clinical Psychology*, 1950, 6(4):343–347.
- [55] Gottschalk L A, Gleser G C, Daniels R S, et al. The Speech Patterns of Schizophrenic Patients: A Method of Assessing Relative Degree of Personal Disorganization and Social Alienation[J]. *The Journal of Nervous and Mental Disease*, 1958, 127(2):153–166.
- [56] Gottschalk L A, Gleser G C. *The Measurement of Psychological States through the Content Analysis of Verbal Behavior*[M]. Univ of California Press, 2022.

- [57] Gottschalk L A, Bechtel R J. Computerized Content Analysis of Natural Language[J]. *Artificial Intelligence in Medicine*, 1989, 1(3):131–137.
- [58] Stone P J, Bales R F, Namenwirth J Z, et al. The general inquirer: A Computer System for Content Analysis and Retrieval based on the Sentence as a Unit of Information[J]. *Behavioral Science*, 1962, 7(4):484.
- [59] Rosenberg S D, Tucker G J. Verbal Behavior and Schizophrenia: The Semantic Dimension[J]. *Archives of General Psychiatry*, 1979, 36(12):1331–1337.
- [60] Weintraub W. *Verbal Behavior: Adaptation and Psychopathology*[M]. Springer Publishing Company, 1981.
- [61] Weintraub W. *Verbal Behavior in Everyday Life*[M]. Springer, 1989.
- [62] Mergenthaler E. Emotion-abstraction Patterns in Verbatim Protocols: A New Way of Describing Psychotherapeutic Processes[J]. *Journal of Consulting and Clinical Psychology*, 1996, 64(6):1306.
- [63] Oxman T E, Rosenberg S D, Tucker G J. The Language of Paranoia[J]. *The American Journal of Psychiatry*, 1982, 139(3):275–282.
- [64] Pennebaker J W, Beall S K. Confronting a Traumatic Event: Toward an Understanding of Inhibition and Disease[J]. *Journal of Abnormal Psychology*, 1986, 95(3):274.
- [65] Pennebaker J W, Francis M E, Booth R J. *Linguistic Inquiry and Word Count (LIWC): A Computerized Text Analysis Program*[M]. Mahwah, NJ: Erlbaum, 2001.
- [66] Rude S, Gortner E M, Pennebaker J. Language Use of Depressed and Depression-vulnerable College Students[J]. *Cognition & Emotion*, 2004, 18(8):1121–1133.
- [67] Pennebaker J W, Chung C K, Ireland M, et al. *The Development and Psychometric Properties of LIWC2007*[M]. Citeseer, 2007.
- [68] Ramirez-Esparza N, Chung C, Kacewic E, et al. The Psychology of Word Use in Depression Forums in English and in Spanish: Testing Two Text Analytic Approaches[C]. *Proceedings of the International AAAI Conference on Web and Social Media*. 2008. 102–108.
- [69] Tausczik Y R, Pennebaker J W. The Psychological Meaning of Words: LIWC and Computerized Text Analysis Methods[J]. *Journal of Language and Social Psychology*, 2010, 29(1):24–54.
- [70] Chung C, Pennebaker J. The Psychological Functions of Function Words[C]. *Proceedings of Social Communication*. Psychology Press, 2011, 2011. 343–359.
- [71] Moreno M A, Jelenchick L A, Egan K G, et al. Feeling Bad on Facebook: Depression Disclosures by College Students on a Social Networking Site[J]. *Depression and Anxiety*, 2011, 28(6):447–455.
- [72] Tsugawa S, Kikuchi Y, Kishino F, et al. Recognizing Depression from Twitter Activity[C]. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 2015. 3187–3196.
- [73] Miller G A, Beckwith R, Fellbaum C, et al. Introduction to WordNet: An On-line Lexical Database[J]. *International Journal of Lexicography*, 1990, 3(4):235–244.
- [74] Dong Z, Dong Q. HowNet-a Hybrid Language and Knowledge Resource[C]. *International Conference on Natural Language Processing and Knowledge Engineering, 2003. Proceedings*. 2003. 2003. 820–824.
- [75] Sparck Jones K. A Statistical Interpretation of Term Specificity and its Application in Retrieval[J]. *Journal of Documentation*, 1972, 28(1):11–21.

- [76] Leiva V, Freire A. Towards Suicide Prevention: Early Detection of Depression on Social Media[C]. *Internet Science: 4th International Conference, INSCI 2017, Thessaloniki, Greece, November 22-24, 2017, Proceedings 4*. 2017. 428–436.
- [77] Stankevich M, Isakov V, Devyatkin D, et al. Feature Engineering for Depression Detection in Social Media[C]. *ICPRAM*. 2018. 426–431.
- [78] Ku L W, Chen H H. Mining Opinions from the Web: Beyond Relevance Retrieval[J]. *Journal of the American Society for Information Science and Technology*, 2007, 58(12):1838–1850.
- [79] 李家俊. 基于多特征加权和混合网络的文本情感分类算法研究 [D]. 西南交通大学, 2021.
- [80] 王婷, 杨文忠. 文本情感分析方法研究综述 [J]. 计算机工程与应用, 2021, 57(12):11–24.
- [81] Wang X, Zhang C, Ji Y, et al. A Depression Detection Model based on Sentiment Analysis in Micro-blog Social Network[C]. *Trends and Applications in Knowledge Discovery and Data Mining: PAKDD 2013 International Workshops*. 2013. 201–213.
- [82] Cambria E, Speer R, Havasi C, et al. SenticNet: A Publicly Available Semantic Resource for Opinion Mining[J]. *AAAI Fall Symposium - Technical Report*, 2010. 14–18.
- [83] Vaswani A, Shazeer N, Parmar N, et al. Attention is All You Need[C]. *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017. 6000–6010.
- [84] Devlin J, Chang M W, Lee K, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C]. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019. 4171–4186.
- [85] Orabi A H, Buddhitha P, Orabi M H, et al. Deep Learning for Depression Detection of Twitter Users[C]. *Proceedings of the Fifth Workshop on Computational Linguistics and Clinical Psychology: From Keyboard to Clinic*. 2018. 88–97.
- [86] Zogan H, Razzak I, Jameel S, et al. DepressionNet: Learning Multi-modalities with User Post Summarization for Depression Detection on Social Media[C]. *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2021. 133–142.
- [87] Haque A, Reddi V, Gialanza T. Deep Learning for Suicide and Depression Identification with Unsupervised Label Correction[C]. *Artificial Neural Networks and Machine Learning–ICANN 2021: 30th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 14–17, 2021, Proceedings, Part V 30*. 2021. 436–447.
- [88] Gu A, Dao T. Mamba: Linear-time Sequence Modeling with Selective State Spaces[Z/OL]. *ArXiv Preprint ArXiv:2312.00752*, 2023. <https://arxiv.org/abs/2312.00752>.
- [89] Poświatka R, Perełkiewicz M. OPI@ LT-EDI-ACL2022: Detecting Signs of Depression from Social Media Text using RoBERTa Pre-trained Language Models[C]. *Proceedings of the Second Workshop on Language Technology for Equality, Diversity and Inclusion*. 2022. 276–282.
- [90] Guo D, Yang D, Zhang H, et al. Deepseek-r1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning[Z/OL]. *ArXiv Preprint ArXiv:2501.12948*, 2025. <https://arxiv.org/abs/2501.12948>.
- [91] 杨璐璐. 基于可穿戴式传感网络的人体异常行为识别 [D]. 南京邮电大学, 2015.

- [92] 合肥工业大学. 基于交叉小波的 SCA-SVM 电机滚动轴承的故障诊断方法 [P]. CN:202210452154.9, 2024-03-22.
- [93] 李笑雪, 张思凝, 王娅娅, 等. 基于树型学习算法的短期流量预测研究 [J]. 电子测试, 2021, (19):48–50.
- [94] 尹卓, 许甜, 陈阳舟, 等. 基于集成聚类的交叉口车辆行驶路径提取方法 [J]. 计算机应用与软件, 2020, 37(07):180–187+193.
- [95] LeCun Y, Boser B, Denker J, et al. Handwritten Digit Recognition with a Back-propagation Network[J]. *Advances in Neural Information Processing Systems*, 1989, 2:396–404.
- [96] Kim Y. Convolutional Neural Networks for Sentence Classification[C]. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. 1746–1751.
- [97] Hochreiter S, Schmidhuber J. Long Short-term Memory[J]. *Neural Computation*, 1997, 9(8):1735–1780.
- [98] Graves A, Schmidhuber J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures[J]. *Neural networks*, 2005, 18(5-6):602–610.
- [99] Sun A. 结巴中文分词 [CP/OL]. <https://github.com/fxsjy/jieba>.
- [100] 裴智勇. 蛋白质相互作用规律及预测方法研究 [D]. 内蒙古科技大学, 2010.
- [101] 刘振宇. 基于语音的抑郁识别方法及关键技术研究 [D]. 兰州大学, 2017.
- [102] Zou B, Han J, Wang Y, et al. Semi-structural Interview-based Chinese Multimodal Depression Corpus towards Automatic Preliminary Screening of Depressive Disorders[J]. *IEEE Transactions on Affective Computing*, 2022, 14(4):2823–2838.
- [103] 龚栩, 黄宇霞, 王妍, 等. 中国面孔表情图片系统的修订 [J]. 中国心理卫生杂志, 2011, 25(01):40–46.
- [104] Boersma P, Van Heuven V. Speak and Unspeak with PRAAT[J]. *Glot International*, 2001, 5(9/10):341–347.
- [105] IBM, Microsoft. Multimedia Programming Interface and Data Specifications 1.0[R/OL]. 1991. <https://www.aelius.com/njh/wavemetatools/doc/riffmci.pdf>.
- [106] 熊子瑜. Praat 语音软件使用手册 [A]. 中国社会科学院语言所语音研究室, 2004.
- [107] Sheehan D V, Leclerc Y, Sheehan K H, et al. The Mini-International Neuropsychiatric Interview (MINI): The Development and Validation of a Structured Diagnostic Psychiatric Interview for DSM-IV and ICD-10[J]. *Journal of Clinical Psychiatry*, 1998, 59(20):22–33.
- [108] Sun Y, Wang S, Feng S, et al. ERNIE 3.0: Large-scale Knowledge Enhanced Pre-training for Language Understanding and Generation[Z/OL]. *ArXiv Preprint ArXiv:2107.02137*, 2021. <https://arxiv.org/abs/2107.02137>.
- [109] Cui Y, Che W, Liu T, et al. Pre-training with Whole Word Masking for Chinese BERT[J]. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 2021, 29:3504–3514.
- [110] Wang Q, Dai S, Xu B, et al. Building Chinese Biomedical Language Models via Multi-level Text Discrimination[Z/OL]. *ArXiv Preprint ArXiv:2110.07244*, 2021. <https://arxiv.org/abs/2110.07244>.
- [111] Hu J, Shen L, Sun G. Squeeze-and-excitation Networks[C]. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018. 7132–7141.

- [112] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition[C]. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016. 770–778.
- [113] Sabour S, Frosst N, Hinton G E. Dynamic Routing between Capsules[J]. *Advances in Neural Information Processing Systems*, 2017, 30:3859–3869.
- [114] Ghosh T, Al Banna M H, Al Nahian M J, et al. An Attention-based Hybrid Architecture with Explainability for Depressive Social Media Text Detection in Bangla[J]. *Expert Systems with Applications*, 2023, 213:119007.
- [115] Thekkekara J P, Yongchareon S, Liesaputra V. An Attention-based CNN-BiLSTM Model for Depression Detection on Social Media Text[J]. *Expert Systems with Applications*, 2024, 249:123834.
- [116] Lin Y, Meng Y, Sun X, et al. BertGCN: Transductive Text Classification by Combining GNN and BERT[C]. *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Association for Computational Linguistics, 2021. 1456–1462.
- [117] Ilias L, Mouzakitis S, Askounis D. Calibration of Transformer-based Models for Identifying Stress and Depression in Social Media[J]. *IEEE Transactions on Computational Social Systems*, 2023, 11(2):1979–1990.
- [118] 郝超, 裴杭萍, 孙毅. 融合 BERT 和图注意力网络的多标签文本分类 [J]. 计算机系统应用, 2022, 31(06):167–174.
- [119] Selim M M, Assiri M S. Enhancing Arabic Text-to-speech Synthesis for Emotional Expression in Visually Impaired Individuals using the Artificial Hummingbird and Hybrid Deep Learning Model[J]. *Alexandria Engineering Journal*, 2025, 119:493–502.
- [120] 录鹏东. 基于语音信号的样本稀疏近似抑郁识别研究 [D]. 兰州大学, 2021.
- [121] 邢玉娟. 多情绪刺激下基于语音信号的抑郁识别关键技术研究 [D]. 兰州大学, 2023.
- [122] Van der Maaten L, Hinton G. Visualizing Data using t-SNE.[J]. *Journal of Machine Learning Research*, 2008, 9(11).
- [123] Fan R E, Chang K W, Hsieh C J, et al. LIBLINEAR: A Library for Large Linear Classification[J]. *The Journal of Machine Learning Research*, 2008, 9:1871–1874.
- [124] Lundberg S M, Lee S I. A Unified Approach to Interpreting Model Predictions[C]. *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017. 4768–4777.

附录

A. 1 抑郁症文本识别系统设计

本研究围绕基于轻量化 Web 框架构建的抑郁症智能分析平台展开，重点阐述如何将理论模型转化为实际应用服务。该平台通过集成语义解析、风险预测和可视化功能模块，为心理健康服务的数字化转型提供可扩展的技术支持。

A. 1. 1 需求分析

在需求分析方面，本研究从功能需求、性能要求和专业适配三个维度构建分析框架。功能需求方面，平台需要实现三方面核心能力：第一，建立支持多源异构数据的处理体系，整合社交媒体文本、临床访谈录音等不同形态数据，形成包含数据清洗和语义标注的标准化处理流程；第二，构建融合传统机器学习与深度学习的混合模型，通过分词处理、词性标注与语义特征提取模块的协同工作，确保抑郁倾向识别的准确率达到 75% 以上；第三，开发具备隐私保护功能的交互式 Web 应用，实现实时文本分析、风险图谱展示和三级预警功能，用户操作响应时间需控制在 3 秒以内。

系统性能设计需着重解决两个关键问题：服务承载能力和架构扩展性。采用轻量化框架与分布式存储技术，确保系统每日可处理上万条文本数据，满足医疗信息安全三级等保要求。通过模块化设计支持算法组件的灵活替换，同时优化跨平台适配性以兼容各类终端设备。数据可视化部分需包含情绪变化趋势图、风险热力图等动态交互组件，支持多维度的分析结果展示。

针对心理健康领域的特殊性，平台设计强调三个专业适配原则：临床合规性方面，基于 DSM-5 诊断标准构建知识图谱，实现临床规则与机器学习模型的有机融合；技术难点方面，重点解决中文隐喻识别和情感分析问题，建立三级风险预警机制并配套标准化干预方案；系统扩展性方面，预留标准化数据接口，支持与医院 HIS 系统对接，构建开放式的心理健康服务生态系统。

通过系统化的需求分析框架构建，本研究在确保平台技术先进性的同时，增强了其在临床实践中的适用性，为后续的系统设计与开发工作提供了理论依据。该分析框架的创新之处在于将技术实现路径、服务效能指标与专业适配要求进行有机结合，为同类心理健康分析系统的研发提供了可参考的范式。

A. 1. 2 系统设计方案

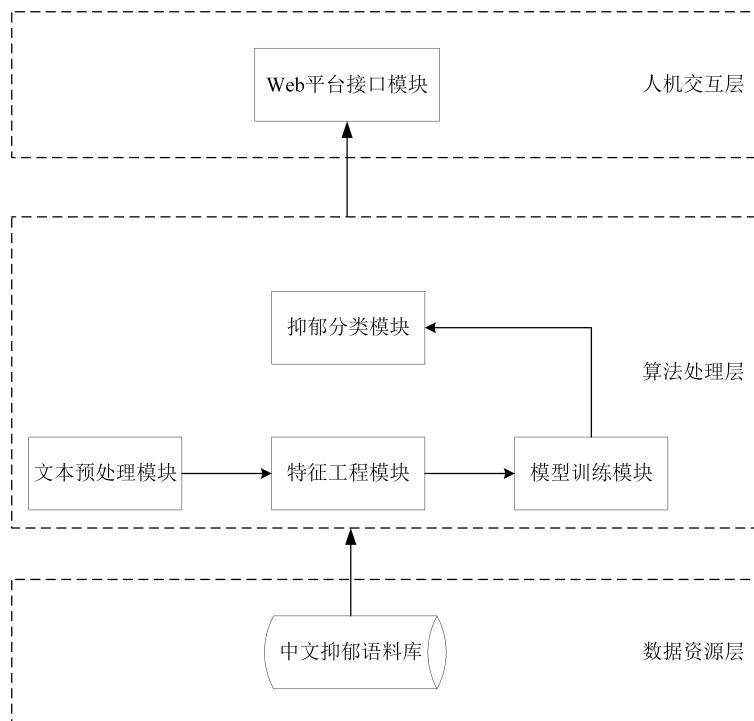


图 A.1 抑郁症文本识别系统设计方案

基于中文文本的抑郁识别系统旨在对用户输入的文本内容进行智能化分析并输出抑郁倾向判定结果，本文据此构建分层系统架构（如图A.1所示）。本系统采用轻量化的三层架构设计，各层功能明确且解耦合。数据层集成 ERNIE-3.0 预训练语言模型，构建动态特征提取管道，支持在线文本向量化处理；算法层基于 DepHybrid 框架，融合 ERNIE 语义表征与 DepHybrid 分类器（代码中为 Joblib 持久化模型）；交互层采用 Streamlit 框架实现单脚本全栈开发，通过组件化设计构建响应式界面。

图A.2展示了系统分层架构的组件交互关系，箭头表示数据/控制流方向。图A.3从静态结构视角呈现了领域模型的设计，定义了实体类、控制类与边界类的分层职责。图A.4描述了用户交互的核心业务流程，包含条件判断与异常处理逻辑。图A.5通过用例图明确了系统功能边界与角色参与关系。

该抑郁倾向分析系统基于分层架构设计，包含数据层、算法层与交互层三大模块。数据层整合 ERNIE-3.0 预训练模型和抑郁症语料库，算法层通过特征提取器将文本转化为词向量，并由 DepHybrid 分类器进行预测，交互层通过 Streamlit 实现用户界面与可视化报告渲染。用户输入文本后，系统触发 ERNIE 特征提取和混合模型预测，根据抑郁概率生成红色警报或绿色健康标识，最终输出可视化分析报告；若未检测到输入则提示重新提交。运行时前端通过请求路由协调文本预处理、特征工程和模型预测流程，结合 ERNIE 服务和模型缓存加速处理，并记录预测日志至语料库。代码层面通过 DepressionClassifier 类封装模型加载与预测方法，FeatureExtractor 处理文本分词与嵌入生成，StreamlitApp 类管理前端交互，三类协作实现从文本输入到结果展示的全流程。

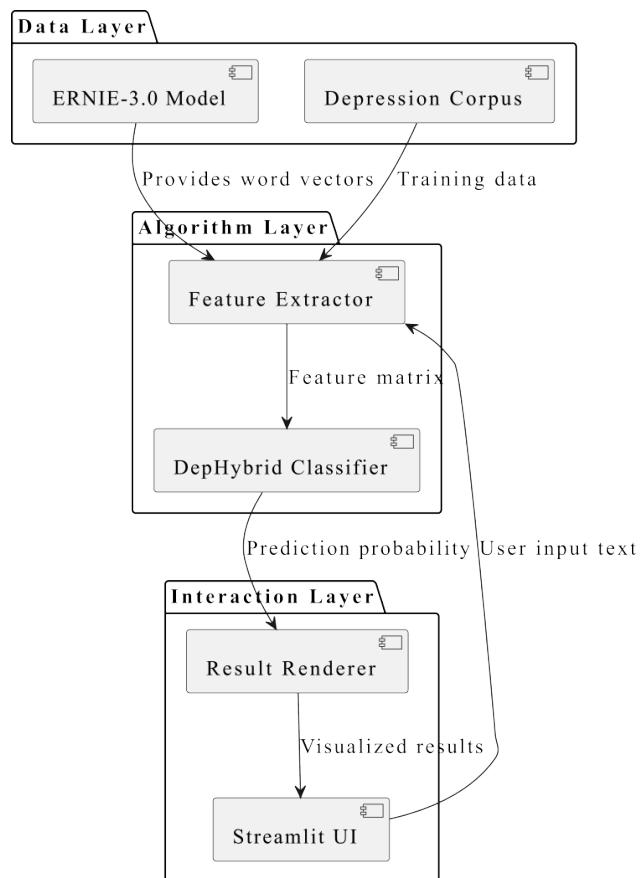


图 A.2 抑郁症文本识别系统组件图

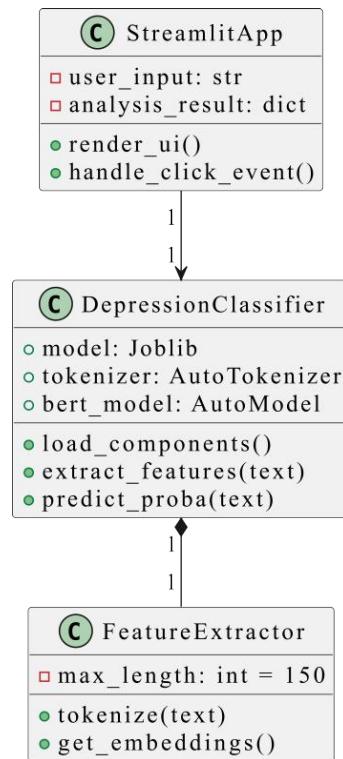


图 A.3 抑郁症文本识别系统类图

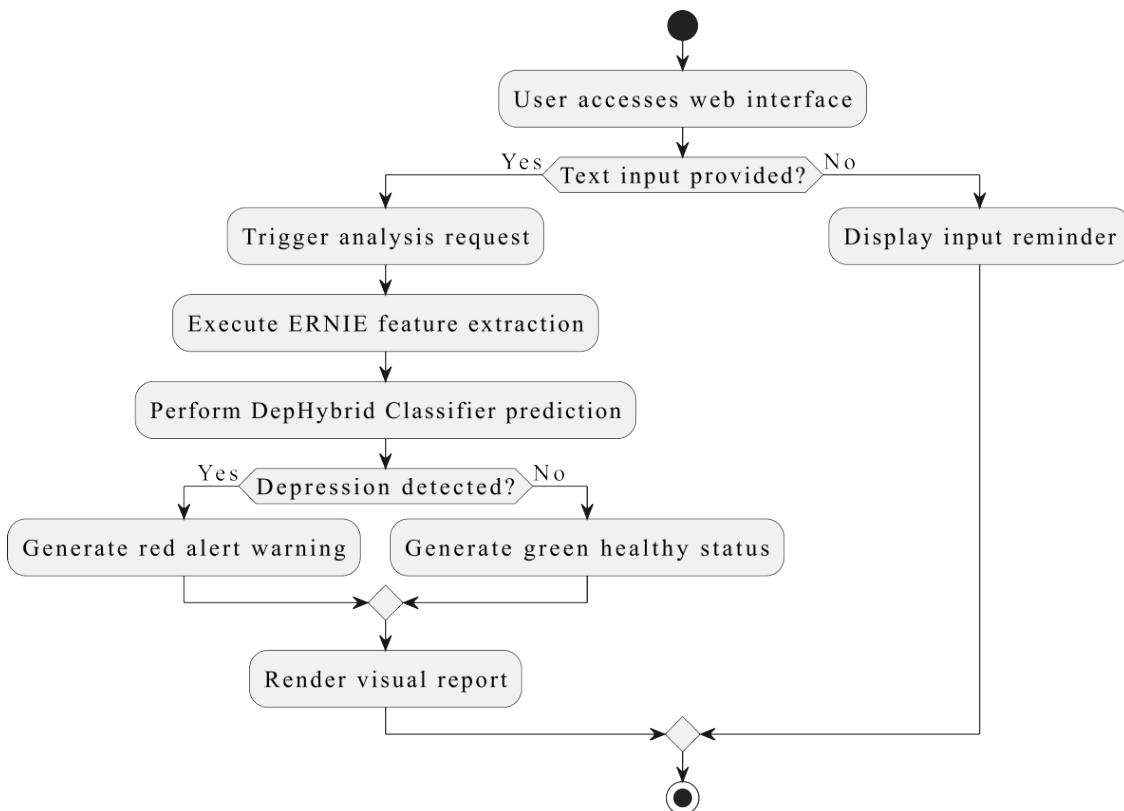


图 A.4 抑郁症文本识别系统活动图

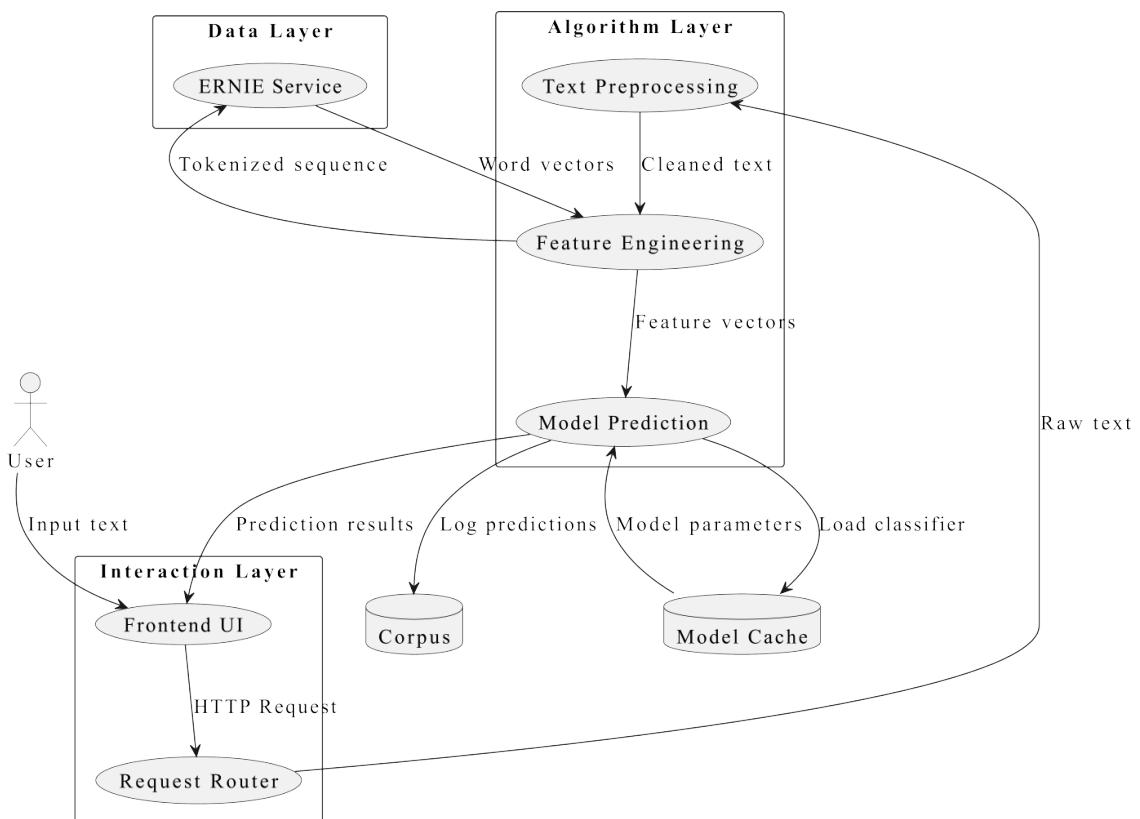


图 A.5 抑郁症文本识别系统用例图

A. 1. 3 系统展示

为了大致了解系统的状况，本文基于 Streamlit 框架进行开发，较其他框架而言，Streamlit 让开发者完全用 Python 构建 UI，减少了全栈开发的复杂度，而使用 Flask 或 Django 需要更多的前后端集成工作，需要处理 HTML/CSS/JavaScript，深入学习 Web 开发。

系统的核心模块 Web 平台接口的人机交互界面如图A.6，可以看到网页主要由侧边栏与文本框组成，侧边栏提供使用说明及注意事项，文本框则供用户输入相应文本，检测是否存在抑郁倾向；当输入待检测文本后（如“最近我总是感到情绪低落，对任何事情都提不起兴趣...”），点击“开始分析”按钮，得到如图A.7的结果，并给出了相关建议；拖动右侧的滚动条，可以看到页面底部有一个“查看详细分析数据”的下拉菜单（图A.8），点击菜单可以看到特征维度及预测概率（图A.9）。对于左侧的侧边栏，可以点击菜单按钮让其隐藏（图A.10）。



图 A.6 抑郁症文本识别系统界面



图 A.7 系统分析界面

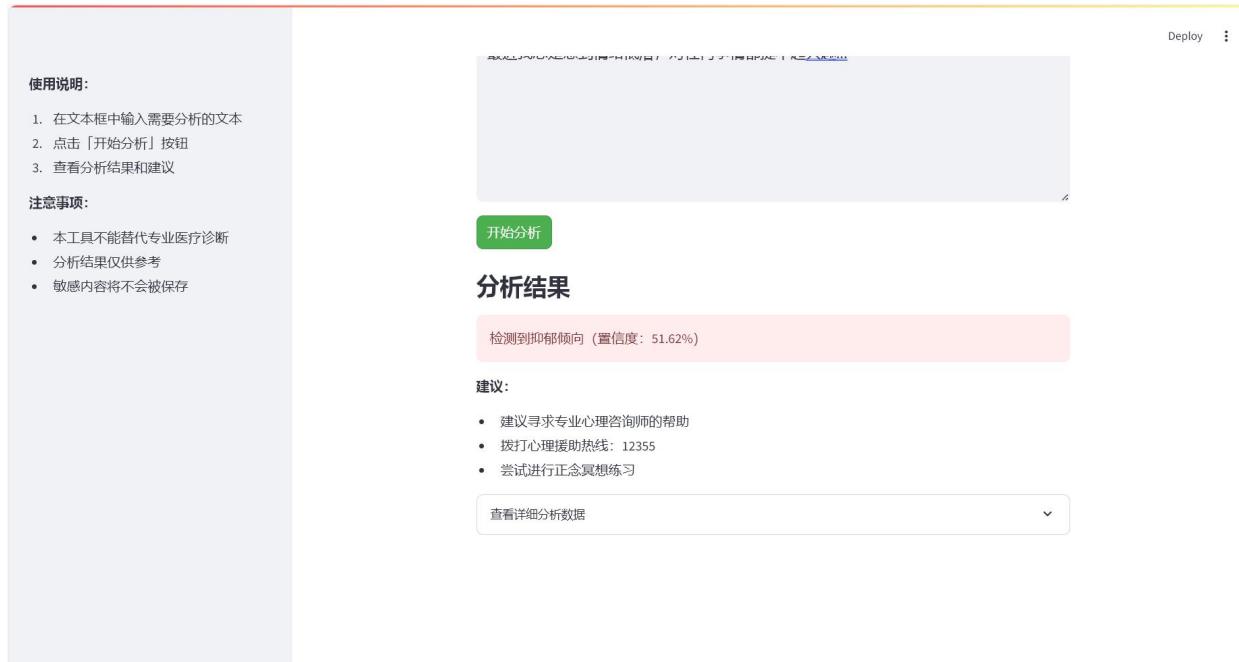


图 A.8 系统底部分析界面



图 A.9 系统详细分析数据界面



图 A.10 系统侧边栏隐藏后界面

A.1.4 系统设计小结

本研究基于 Streamlit 框架成功构建了抑郁症文本智能分析系统，实现了从用户输入到抑郁倾向判定的端到端服务。相较于传统 Web 框架，Streamlit 的 Python 全栈开发特性显著降低了交互界面构建复杂度，通过侧边栏导航、多级结果展示（文本输入、实时预测、特征维度可视化）的模块化设计，在保证功能完整性的同时提高了开发效率。系统采用“输入-分析-解释”的闭环交互逻辑，通过案例验证表

明能够有效解析典型抑郁表述（如情绪低落、兴趣缺失等），并提供分级建议，为心理健康服务的轻量化部署提供了可行方案。

本系统目前存在三方面待改进领域：（1）系统需提升移动端界面兼容性与可视化呈现形式多样性；（2）单纯依赖文本信息的评估模式难以全面捕捉用户心理特征；（3）系统对隐喻性表述及地域性语言变体的解析准确性仍需强化。因此后续技术演进将聚焦四大核心方向：（1）系统融合语音语调、微表情等多维度生物特征构建综合分析模型；（2）系统集成大语言模型提升语境感知能力，同时搭建可迭代更新的跨文化语义知识图谱；（3）系统运用组件化前端框架实现自适应多端交互界面；（4）系统通过联邦学习等隐私保护技术与医疗系统建立数据协作机制。

A. 2 实验环境配置

A. 2. 1 Windows 系统

```
1 catboost==1.2.2
2 gensim==4.3.3
3 jieba==0.42.1
4 joblib==1.3.2
5 keras==2.10.0
6 lightgbm==3.3.5
7 matplotlib==3.7.1
8 numpy==1.24.4
9 pandas==1.5.3
10 pillow==9.4.0
11 scikit_learn==1.3.2
12 seaborn==0.12.2
13 shap==0.47.2
14 # streamlit==1.45.0
15 tensorflow==2.10.1
16 tensorflow_intel==2.10.1
17 torch==2.7.0
18 torch_geometric==2.6.1
19 tqdm==4.64.1
20 transformers==4.36.2
21 wordcloud==1.9.4
22 xgboost==1.7.3
```

A. 2. 2 Ubuntu 子系统

```
1 auto_sklearn==0.15.0
2 jieba==0.42.1
3 numpy==1.21.0
4 pandas==1.3.5
5 scikit_learn==0.24.2
```

A. 3 实验核心源代码

A. 3. 1 DepHybrid 抑郁识别架构

```
1 """
2 ERNIE-based Hybrid Model for Text Classification with Stratified Cross-Validation
3 This script implements a hybrid model combining logistic regression and a custom ANN with attention mechanisms,
4 using ERNIE embeddings for Chinese text classification.
5 """
6
7 import numpy as np
8 import pandas as pd
9 import tensorflow as tf
10 tf.config.set_visible_devices([], 'GPU') # Disable GPU for TensorFlow
11 import os
12 os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com/' # Set HuggingFace mirror
13 import torch
14 import random
15 from transformers import AutoTokenizer, AutoModel
16 from sklearn.model_selection import StratifiedKFold
17 from sklearn.linear_model import LogisticRegression
18 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
19 import keras.backend as K
20 from keras.models import Model
21 from keras.layers import Input, Bidirectional, LSTM, Dense, Dropout, Layer, GlobalAveragePooling1D, Multiply,
22     Reshape, Flatten, LayerNormalization
23 from keras.optimizers import Adam
24 from keras.callbacks import EarlyStopping
25
26 # Set random seeds for reproducibility
27 tf.random.set_seed(42)
28 random.seed(42)
29 np.random.seed(42)
30
31 # Load dataset
32 data = pd.read_excel("./Dataset.xlsx")
33
34 # Initialize ERNIE tokenizer and model for Chinese text processing
35 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
36 model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
37
38 def extract_word_embeddings(text, tokenizer, model, max_length=100):
39     """
40         Extract word-level embeddings using ERNIE model.
41
42     Args:
43         text (str): Input text
44         tokenizer: ERNIE tokenizer
45         model: ERNIE model
46         max_length (int): Maximum sequence length
47
48     Returns:
49         numpy.ndarray: Word embeddings with shape (seq_length, hidden_size)
50         """
51     inputs = tokenizer(text, return_tensors='pt',
52                       max_length=max_length,
53                       padding='max_length',
```

```

53             truncation=True)
54
55     with torch.no_grad():
56         outputs = model(**inputs)
57
58     return outputs.last_hidden_state.squeeze().cpu().numpy()
59
60 def extract_sentence_embeddings(text, tokenizer, model, max_length=100):
61     """
62     Extract sentence-level embeddings using mean pooling.
63
64     Args:
65         text (str): Input text
66         tokenizer: ERNIE tokenizer
67         model: ERNIE model
68         max_length (int): Maximum sequence length
69
70     Returns:
71         numpy.ndarray: Sentence embedding with shape (hidden_size,)
72     """
73     inputs = tokenizer(text, return_tensors='pt',
74                         max_length=max_length,
75                         padding=True,
76                         truncation=True)
77
78     with torch.no_grad():
79         outputs = model(**inputs)
80
81     return outputs.last_hidden_state.mean(dim=1).squeeze().cpu().numpy()
82
83 # Generate embeddings for all texts
84 X_word = np.array([extract_word_embeddings(text, tokenizer, model, 150) for text in data['text']])
85 X_sent = np.array([extract_sentence_embeddings(text, tokenizer, model, 150) for text in data['text']])
86 y = data['class'].values # Target labels
87
88 class SelfAttention(Layer):
89     """Implementation of self-attention mechanism with learnable weights"""
90
91     def __init__(self, units=128, **kwargs):
92         super().__init__(**kwargs)
93         self.units = units
94
95     def build(self, input_shape):
96         # Initialize weight matrices for query, key and value
97         self.W_q = self.add_weight(name='W_q',
98                                     shape=(input_shape[-1], self.units),
99                                     initializer='glorot_uniform')
100        self.W_k = self.add_weight(name='W_k',
101                                     shape=(input_shape[-1], self.units),
102                                     initializer='glorot_uniform')
103        self.W_v = self.add_weight(name='W_v',
104                                     shape=(input_shape[-1], self.units),
105                                     initializer='glorot_uniform')

```

```
106     super().build(input_shape)
107
108     def call(self, x):
109         # Compute query, key, value projections
110         q = K.dot(x, self.W_q) # [batch_size, seq_len, units]
111         k = K.dot(x, self.W_k)
112         v = K.dot(x, self.W_v)
113
114         # Calculate attention scores
115         scores = K.batch_dot(q, K.permute_dimensions(k, (0, 2, 1)))
116         scores = K.softmax(scores / K.sqrt(K.cast(self.units, 'float32')))
117
118         # Apply attention weights to values
119         return K.batch_dot(scores, v)
120
121     def compute_output_shape(self, input_shape):
122         return (input_shape[0], input_shape[1], self.units)
123
124     def channel_attention(x):
125         """Channel attention mechanism for feature recalibration"""
126         gap = GlobalAveragePooling1D()(x)
127         gap = Dense(x.shape[-1]//8, activation='relu')(gap)
128         gap = Dense(x.shape[-1], activation='sigmoid')(gap)
129         return Multiply()([x, gap])
130
131     def build_hybrid_model(input_shape):
132         """
133             Construct hybrid neural network architecture with:
134             - BiLSTM layers
135             - Self-attention mechanism
136             - Channel attention
137         """
138         inputs = Input(shape=input_shape)
139
140         # BiLSTM layer with sequence output
141         x = Bidirectional(LSTM(64, return_sequences=True))(inputs)
142
143         # Self-attention with layer normalization
144         x = SelfAttention(units=64)(x)
145         x = LayerNormalization()(x)
146
147         # Second BiLSTM layer with vector output
148         x = Bidirectional(LSTM(64))(x)
149
150         # Dense layers with dropout
151         x = Dense(64, activation='relu')(x)
152         x = Dropout(0.3)(x)
153         x = Dense(32, activation='relu')(x)
154
155         # Channel attention mechanism
156         x = Reshape((32, 1))(x)
157         x = channel_attention(x)
158         x = Flatten()(x)
```

```
159
160     # Output layer
161     outputs = Dense(1, activation='sigmoid')(x)
162
163     model = Model(inputs=inputs, outputs=outputs)
164     optimizer = Adam(learning_rate=1e-3)
165     model.compile(loss='binary_crossentropy',
166                     optimizer=optimizer,
167                     metrics=['accuracy'])
168     return model
169
170 # Training configuration
171 BATCH_SIZE = 32
172 EPOCHS = 100
173 kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
174
175 # Metrics storage
176 metrics = {
177     'accuracy': [],
178     'precision': [],
179     'recall': [],
180     'f1': [],
181     'auc': []
182 }
183
184 for train_idx, val_idx in kfold.split(X_word, y):
185     # Split data for current fold
186     X_train, X_val = X_word[train_idx], X_word[val_idx]
187     y_train, y_val = y[train_idx], y[val_idx]
188     X_sent_train, X_sent_val = X_sent[train_idx], X_sent[val_idx]
189
190     # ---- Logistic Regression Model ---
191     lr_model = LogisticRegression(max_iter=1000, random_state=42)
192     lr_model.fit(X_sent_train, y_train)
193     lr_probs = lr_model.predict_proba(X_sent_val)[:, 1] # Positive class probabilities
194
195     # ---- Neural Network Model ---
196     ann_model = build_hybrid_model((X_train.shape[1], X_train.shape[2]))
197     ann_model.fit(X_train, y_train,
198                   batch_size=BATCH_SIZE,
199                   epochs=EPOCHS,
200                   validation_data=(X_val, y_val),
201                   verbose=0,
202                   callbacks=[EarlyStopping(patience=10, restore_best_weights=True)])
203     ann_probs = ann_model.predict(X_val, verbose=0).ravel()
204
205     # ---- Ensemble with Equal Weighting ---
206     ensemble_probs = 0.5 * lr_probs + 0.5 * ann_probs
207     y_pred = (ensemble_probs > 0.5).astype(int)
208
209     # Calculate metrics
210     metrics['accuracy'].append(accuracy_score(y_val, y_pred))
211     metrics['precision'].append(precision_score(y_val, y_pred))
```

```

212     metrics['recall'].append(recall_score(y_val, y_pred))
213     metrics['f1'].append(f1_score(y_val, y_pred))
214     metrics['auc'].append(roc_auc_score(y_val, y_pred))
215
216     # Calculate and display final metrics
217     print(f"Cross-Validation Results (Mean ± SD):")
218     print(f"Accuracy: {np.mean(metrics['accuracy']):.4f}±{np.std(metrics['accuracy']):.4f}")
219     print(f"Precision: {np.mean(metrics['precision']):.4f}±{np.std(metrics['precision']):.4f}")
220     print(f"Recall: {np.mean(metrics['recall']):.4f}±{np.std(metrics['recall']):.4f}")
221     print(f"F1 Score: {np.mean(metrics['f1']):.4f}±{np.std(metrics['f1']):.4f}")
222     print(f"AUC-ROC: {np.mean(metrics['auc']):.4f}±{np.std(metrics['auc']):.4f}")

```

A. 3. 2 基于 ERNIE-LR 的 UCB 参数优化方法与其他优化方法的对比实验

```

1 """
2 ERNIE-based Text Classification with Hyperparameter Tuning Comparison
3 This script compares three hyperparameter tuning methods (Reinforcement Learning UCB, Grid Search, Random Search
4 )
5 for Logistic Regression on ERNIE embeddings using stratified 10-fold cross-validation.
6 """
7
8 import numpy as np
9 import pandas as pd
10 import tensorflow as tf
11 import os
12 os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com/' # Use huggingface mirror
13 import torch
14 import random
15 from transformers import AutoTokenizer, AutoModel
16 from sklearn.model_selection import StratifiedKFold
17 from sklearn.linear_model import LogisticRegression
18 from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, roc_auc_score
19 import math
20 from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
21 from scipy.stats import loguniform
22
23 # Set all random seeds for reproducibility
24 SEED = 42
25 os.environ['PYTHONHASHSEED'] = str(SEED)
26 random.seed(SEED)
27 np.random.seed(SEED)
28 tf.random.set_seed(SEED)
29 os.environ['TF_DETERMINISTIC_OPS'] = '1' # Ensure deterministic GPU operations
30 os.environ['TF_CUDNN_DETERMINISTIC'] = '1'
31
32 # Read dataset
33 data = pd.read_excel("./Dataset.xlsx")
34
35 # Initialize ERNIE model components
36 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
37 model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
38
39 def extract_bert_embeddings(text, tokenizer, model, max_length=100):

```

```

39     """
40     Generate sentence embeddings using ERNIE model
41     Args:
42         text (str): Input text
43         tokenizer: Pretrained tokenizer
44         model: Pretrained ERNIE model
45         max_length (int): Maximum sequence length
46     Returns:
47         numpy.ndarray: Sentence embedding vector
48     """
49     inputs = tokenizer(text, return_tensors='pt', max_length=max_length, padding=True, truncation=True)
50     with torch.no_grad():
51         outputs = model(**inputs)
52         # Use mean pooling for sentence embedding
53         embeddings = outputs.last_hidden_state.mean(dim=1)
54     return embeddings.squeeze().cpu().numpy()
55
56     # Generate ERNIE embeddings for all texts
57     X_embeddings = np.array([extract_bert_embeddings(text, tokenizer, model, max_length=150) for text in data['text']])
58     y = data['class'].values
59
60     def reinforcement_learning_tuning(X_train, y_train, X_val, y_val, n_episodes=10):
61         """
62             Hyperparameter tuning using Upper Confidence Bound (UCB) reinforcement learning
63             Args:
64                 X_train: Training features
65                 y_train: Training labels
66                 X_val: Validation features
67                 y_val: Validation labels
68                 n_episodes (int): Number of exploration episodes
69             Returns:
70                 dict: Best parameters found
71             """
72             # Define hyperparameter search space
73             C_options = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
74             penalty_options = ['l1', 'l2']
75             actions = [{'C': c, 'penalty': p} for c in C_options for p in penalty_options]
76             n_actions = len(actions)
77             action_stats = [{total_reward: 0.0, count: 0} for _ in range(n_actions)]
78
79             for episode in range(n_episodes):
80                 selected_action_idx = None
81                 max_ucb = -np.inf
82
83                 # UCB selection strategy
84                 for i in range(n_actions):
85                     if action_stats[i][count] == 0: # Prioritize unexplored actions
86                         selected_action_idx = i
87                         break
88                     else:
89                         # Calculate Upper Confidence Bound
90                         avg_reward = action_stats[i]['total_reward'] / action_stats[i]['count']
91                         exploration_term = math.sqrt(2 * math.log(episode+1) / action_stats[i]['count'])

```

```

92         ucb = avg_reward + exploration_term
93         if ucb > max_ucb:
94             max_ucb = ucb
95             selected_action_idx = i
96
97     # Get selected hyperparameters
98     params = actions[selected_action_idx]
99
100    try:
101        # Train model with selected parameters
102        model = LogisticRegression(
103            C=params['C'],
104            penalty=params['penalty'],
105            solver='liblinear',
106            max_iter=1000,
107            random_state=SEED
108        )
109        model.fit(X_train, y_train)
110        y_pred = model.predict(X_val)
111        reward = f1_score(y_val, y_pred) # Use F1-score as reward
112    except:
113        reward = -1 # Penalize invalid parameter combinations
114
115    # Update action statistics
116    action_stats[selected_action_idx]['total_reward'] += reward
117    action_stats[selected_action_idx]['count'] += 1
118
119    # Select best performing action
120    best_idx = np.argmax([(s['total_reward'])/s['count'] if s['count']>0 else -1 for s in action_stats])
121    return actions[best_idx]
122
123 def grid_search_tuning(X_train, y_train):
124     """
125     Hyperparameter tuning using exhaustive grid search
126     Args:
127         X_train: Training features
128         y_train: Training labels
129     Returns:
130         dict: Best parameters found
131     """
132     param_grid = {
133         'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
134         'penalty': ['l1', 'l2']
135     }
136     search = GridSearchCV(
137         estimator=LogisticRegression(solver='liblinear', max_iter=1000, random_state=SEED),
138         param_grid=param_grid,
139         scoring='f1',
140         n_jobs=-1
141     )
142     search.fit(X_train, y_train)
143     return search.best_params_
144

```

```
145 def random_search_tuning(X_train, y_train, n_iter=10):
146     """
147     Hyperparameter tuning using random search
148     Args:
149         X_train: Training features
150         y_train: Training labels
151         n_iter (int): Number of iterations
152     Returns:
153         dict: Best parameters found
154     """
155     param_dist = {
156         'C': loguniform(1e-4, 1e3), # Log-uniform distribution for C
157         'penalty': ['l1', 'l2']
158     }
159     search = RandomizedSearchCV(
160         estimator=LogisticRegression(solver='liblinear', max_iter=1000, random_state=SEED),
161         param_distributions=param_dist,
162         n_iter=n_iter,
163         scoring='f1',
164         n_jobs=-1,
165         random_state=SEED
166     )
167     search.fit(X_train, y_train)
168     return search.best_params_
169
170 # Initialize metrics storage
171 metrics = {
172     'UCB': {'acc': [], 'pre': [], 'rec': [], 'f1': [], 'auc': []},
173     'GridSearch': {'acc': [], 'pre': [], 'rec': [], 'f1': [], 'auc': []},
174     'RandomSearch': {'acc': [], 'pre': [], 'rec': [], 'f1': [], 'auc': []}
175 }
176
177 # Stratified 10-fold cross-validation
178 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=SEED)
179
180 for fold_idx, (train_idx, val_idx) in enumerate(skf.split(X_embeddings, y)):
181     print(f"\nProcessing Fold {fold_idx+1}/10")
182
183 # Data splitting
184 X_train, X_val = X_embeddings[train_idx], X_embeddings[val_idx]
185 y_train, y_val = y[train_idx], y[val_idx]
186
187 # Define tuning methods
188 tuning_methods = {
189     'UCB': lambda X, y: reinforcement_learning_tuning(X, y, X_val, y_val),
190     'GridSearch': grid_search_tuning,
191     'RandomSearch': lambda X, y: random_search_tuning(X, y, n_iter=10)
192 }
193
194 # Evaluate each tuning method
195 for method_name, tuning_func in tuning_methods.items():
196     # Hyperparameter tuning
197     best_params = tuning_func(X_train, y_train)
```

```

198     # Train final model
199     final_model = LogisticRegression(
200         **best_params,
201         solver='liblinear',
202         max_iter=1000,
203         random_state=SEED
204     )
205     final_model.fit(X_train, y_train)
206
207
208     # Model evaluation
209     y_pred = final_model.predict(X_val)
210     y_proba = final_model.predict_proba(X_val)[:, 1]
211
212     # Store metrics
213     metrics[method_name]['acc'].append(accuracy_score(y_val, y_pred))
214     metrics[method_name]['pre'].append(precision_score(y_val, y_pred))
215     metrics[method_name]['rec'].append(recall_score(y_val, y_pred))
216     metrics[method_name]['f1'].append(f1_score(y_val, y_pred))
217     metrics[method_name]['auc'].append(roc_auc_score(y_val, y_proba))
218
219 def print_metrics(metric_dict, method_name):
220     """
221     Print performance metrics with mean and standard deviation
222     Args:
223         metric_dict: Dictionary containing metric lists
224         method_name: Name of the tuning method
225     """
226     print(f"\nPerformance Comparison for {method_name}:")
227     print(f"Average Accuracy: {np.mean(metric_dict['acc']):.4f}±{np.std(metric_dict['acc']):.4f}")
228     print(f"Average Precision: {np.mean(metric_dict['pre']):.4f}±{np.std(metric_dict['pre']):.4f}")
229     print(f"Average Recall: {np.mean(metric_dict['rec']):.4f}±{np.std(metric_dict['rec']):.4f}")
230     print(f"Average F1-Score: {np.mean(metric_dict['f1']):.4f}±{np.std(metric_dict['f1']):.4f}")
231     print(f"Average AUC-ROC: {np.mean(metric_dict['auc']):.4f}±{np.std(metric_dict['auc']):.4f}")
232
233     # Print final results
234     for method in metrics.keys():
235         print_metrics(metrics[method], method)

```

A. 3. 3 Streamlit 抑郁文本识别系统

```

1 import streamlit as st
2 from joblib import load
3 import numpy as np
4 import torch
5 import os
6 os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com/'
7 from transformers import AutoTokenizer, AutoModel
8
9 # Load components with caching
10 @st.cache_resource
11 def load_components():
12     # Load trained classifier model

```

```
13     model = load('depression_classifier.joblib')
14
15     # Load ERNIE model and tokenizer
16     tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
17     bert_model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
18
19     return model, tokenizer, bert_model
20
21 # Feature extraction function
22 def extract_features(text, tokenizer, model):
23     # Tokenize input text
24     inputs = tokenizer(text,
25                         return_tensors='pt',
26                         max_length=150,
27                         padding=True,
28                         truncation=True)
29
30     # Get BERT embeddings
31     with torch.no_grad():
32         outputs = model(**inputs)
33
34     # Return mean-pooled embeddings
35     return outputs.last_hidden_state.mean(dim=1).squeeze().cpu().numpy()
36
37 # Create main interface
38 st.title('N{BRAIN} 抑郁症倾向检测系统')
39 st.markdown("""
40 <style>
41     .stTextArea textarea {font-size: 18px;}
42     .stButton button {background-color: #4CAF50; color: white;}
43 </style>
44 """, unsafe_allow_html=True)
45
46 # Initialize components
47 model, tokenizer, bert_model = load_components()
48
49 # User input section
50 user_input = st.text_area("请输入需要分析的文本（建议200–500字）：",
51                         height=200,
52                         placeholder="例如：最近我总是感到情绪低落，对任何事情都提不起兴趣...")
```

```
53
54 # Analysis button handler
55 if st.button('开始分析'):
56     if user_input:
57         with st.spinner('分析中，请稍候...'):
58             try:
59                 # Extract text features
60                 features = np.array([extract_features(user_input, tokenizer, bert_model)])
61
62                 # Make prediction
63                 prediction = model.predict(features)
64                 proba = model.predict_proba(features)[0][1]
65
```

```

66         # Display results
67         st.subheader("分析结果")
68         if prediction[0] == 1:
69             st.error(f"检测到抑郁倾向 (置信度: {proba:.2%}) ")
70             st.markdown("""
71             **建议: **
72             - 建议寻求专业心理咨询师的帮助
73             - 拨打心理援助热线: 12355
74             - 尝试进行正念冥想练习
75             """)
76         else:
77             st.success(f"未检测到明显抑郁倾向 (置信度: {1-proba:.2%}) ")
78             st.markdown("""
79             **保持心理健康小贴士: **
80             - 保持规律作息
81             - 定期进行体育锻炼
82             - 多与亲友沟通交流
83             """)
84
85         # Show detailed analysis data
86         with st.expander("查看详细分析数据"):
87             st.write(f"特征维度: {features.shape}")
88             st.write(f"预测概率: {proba:.4f}")
89
90         except Exception as e:
91             st.error(f"分析失败: {str(e)}")
92         else:
93             st.warning("请输入需要分析的文本内容!")
94
95     # Sidebar information
96     st.sidebar.markdown("""
97     **使用说明: **
98     1. 在文本框中输入需要分析的文本
99     2. 点击「开始分析」按钮
100    3. 查看分析结果和建议
101
102   **注意事项: **
103   - 本工具不能替代专业医疗诊断
104   - 分析结果仅供参考
105   - 敏感内容将不会被保存
106   """)

```

A.4 基于 ERNIE 语义的实验复现源代码

A.4.1 TextCNN

```

1 import random
2 import pandas as pd
3 import numpy as np
4 import tensorflow as tf
5 tf.config.set_visible_devices([], 'GPU') # Disable GPU usage
6 import os

```

```
7  os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com/' # Set HuggingFace mirror
8  from keras.models import Model
9  from keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, Dense, concatenate, Dropout, Flatten
10 from keras.callbacks import EarlyStopping
11 from sklearn.model_selection import StratifiedKFold
12 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
13 import torch
14 from transformers import AutoTokenizer, AutoModel
15
16 # Set random seeds for reproducibility
17 tf.random.set_seed(42)
18 random.seed(42)
19 np.random.seed(42)
20
21 # Load dataset
22 data = pd.read_excel("./Dataset.xlsx")
23
24 # Load pretrained ERNIE model and tokenizer
25 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
26 model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
27
28 def extract_bert_embeddings(text, tokenizer, model, max_length=100):
29     """
30     Generate sentence embeddings using ERNIE model.
31
32     Args:
33         text (str): Input text
34         tokenizer: Pretrained tokenizer
35         model: Pretrained BERT model
36         max_length (int): Maximum sequence length
37
38     Returns:
39         np.array: Sentence embeddings of shape (max_length, embedding_dim)
40     """
41     inputs = tokenizer(text, return_tensors='pt',
42                         max_length=max_length,
43                         padding='max_length',
44                         truncation=True)
45
46     with torch.no_grad():
47         outputs = model(**inputs)
48
49     embeddings = outputs.last_hidden_state
50     return embeddings.squeeze().cpu().numpy()
51
52 # Generate BERT embeddings for all texts
53 X = np.array([extract_bert_embeddings(text, tokenizer, model, max_length=150) for text in data['text']])
54 y = data['class'].values # Assuming 'class' is the label column
55
56 def build_textcnn(input_shape):
57     """
58     Build TextCNN model architecture.
59
```

```
60     Args:  
61         input_shape (tuple): Shape of input data (timesteps, features)  
62  
63     Returns:  
64         keras.Model: Compiled TextCNN model  
65         """  
66         inputs = Input(shape=input_shape)  
67  
68         # Multi-scale convolutional blocks  
69         conv_blocks = []  
70         for kernel_size in [3, 4, 5]:  
71             conv = Conv1D(128, kernel_size, activation='relu')(inputs)  
72             pool = GlobalMaxPooling1D()(conv)  
73             conv_blocks.append(pool)  
74  
75         # Feature concatenation  
76         merged = concatenate(conv_blocks)  
77  
78         # Classification head  
79         dense = Dense(128, activation='relu')(merged)  
80         dropout = Dropout(0.5)(dense)  
81         outputs = Dense(1, activation='sigmoid')(dropout) # Binary classification  
82  
83         model = Model(inputs=inputs, outputs=outputs)  
84         return model  
85  
86     # Training configuration  
87     BATCH_SIZE = 32  
88     EPOCHS = 100  
89  
90     # Initialize metrics storage  
91     accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []  
92  
93     # 10-fold stratified cross-validation  
94     skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)  
95  
96     for train_idx, val_idx in skf.split(X, y):  
97         X_train, X_val = X[train_idx], X[val_idx]  
98         y_train, y_val = y[train_idx], y[val_idx]  
99  
100    # Build and compile model  
101    textcnn = build_textcnn((X_train.shape[1], X_train.shape[2]))  
102    textcnn.compile(loss='binary_crossentropy',  
103                    optimizer='adam',  
104                    metrics=['accuracy'])  
105  
106    # Train with early stopping  
107    textcnn.fit(X_train, y_train,  
108                batch_size=BATCH_SIZE,  
109                epochs=EPOCHS,  
110                validation_data=(X_val, y_val),  
111                verbose=0,  
112                callbacks=[EarlyStopping(patience=10, restore_best_weights=True)])
```

```

113
114     # Predict and evaluate
115     y_pred = (textcnn.predict(tf.convert_to_tensor(X_val, dtype=tf.float32)) > 0.5).astype(int)
116     accuracies.append(accuracy_score(y_val, y_pred))
117     precisions.append(precision_score(y_val, y_pred))
118     recalls.append(recall_score(y_val, y_pred))
119     f1_scores.append(f1_score(y_val, y_pred))
120     aucs.append(roc_auc_score(y_val, y_pred))
121
122     # Print cross-validation results
123     print(f"Cross-validation Accuracy: {np.mean(accuracies):.4f}±{np.std(accuracies):.4f}")
124     print(f"Cross-validation Precision: {np.mean(precisions):.4f}±{np.std(precisions):.4f}")
125     print(f"Cross-validation Recall: {np.mean(recalls):.4f}±{np.std(recalls):.4f}")
126     print(f"Cross-validation F1-score: {np.mean(f1_scores):.4f}±{np.std(f1_scores):.4f}")
127     print(f"Cross-validation AUC-ROC: {np.mean(aucs):.4f}±{np.std(aucs):.4f}")

```

A. 4. 2 CapsNet

```

1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torch.utils.data import Dataset, DataLoader
5 import numpy as np
6 import pandas as pd
7 import random
8 from sklearn.model_selection import StratifiedKFold
9 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
10 from transformers import AutoTokenizer, AutoModel
11
12     # Set random seeds for reproducibility
13 torch.manual_seed(42)
14 np.random.seed(42)
15 random.seed(42)
16 torch.backends.cudnn.deterministic = True
17
18     # Load dataset
19 data = pd.read_excel("./Dataset.xlsx")
20
21     # Initialize BERT model and tokenizer
22 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
23 bert_model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
24
25 def extract_bert_embeddings(text, tokenizer, model, max_length=100):
26     """Extract BERT embeddings for input text."""
27     inputs = tokenizer(text, return_tensors='pt', max_length=max_length, padding=True, truncation=True)
28     with torch.no_grad():
29         outputs = model(**inputs)
30         # Average embeddings across tokens
31     embeddings = outputs.last_hidden_state.mean(dim=1)
32     return embeddings.squeeze().cpu().numpy()
33
34     # Generate BERT embeddings for all texts
35 X = np.array([extract_bert_embeddings(text, tokenizer, bert_model, max_length=150) for text in data['text']])

```

```

36     y = data['class'].values.astype('int32')
37
38     class TextDataset(Dataset):
39         """Custom PyTorch Dataset for text embeddings and labels."""
40         def __init__(self, features, labels):
41             self.features = features
42             self.labels = labels
43
44         def __len__(self):
45             return len(self.labels)
46
47         def __getitem__(self, idx):
48             return (
49                 torch.tensor(self.features[idx], dtype=torch.float32),
50                 torch.tensor(self.labels[idx], dtype=torch.long)
51             )
52
53     class CapsuleLayer(nn.Module):
54         """Implementation of a capsule layer with dynamic routing."""
55         def __init__(self, num_capsule, dim_capsule, input_num_capsule, input_dim_capsule, routings=3):
56             super().__init__()
57             self.num_capsule = num_capsule # Number of capsules in this layer
58             self.dim_capsule = dim_capsule # Dimension of each capsule
59             self.routings = routings # Number of routing iterations
60
61             # Weight matrix for transforming input capsules
62             self.W = nn.Parameter(
63                 torch.randn(num_capsule, input_num_capsule, input_dim_capsule, dim_capsule)
64
65         def squash(self, vectors, dim=-1):
66             """Non-linear 'squashing' function to ensure short vectors get shrunk."""
67             squared_norm = (vectors ** 2).sum(dim=dim, keepdim=True)
68             scale = squared_norm / (1 + squared_norm) / torch.sqrt(squared_norm + 1e-8)
69             return scale * vectors
70
71         def forward(self, x):
72             batch_size = x.size(0)
73             # Expand input for matrix operations
74             x = x.unsqueeze(1).unsqueeze(-1).repeat(1, self.num_capsule, 1, 1, 1)
75             W = self.W.unsqueeze(0).repeat(batch_size, 1, 1, 1, 1)
76
77             # Transform input capsules (u_hat)
78             u_hat = torch.matmul(x.transpose(3, 4), W).squeeze(3)
79
80             # Dynamic routing algorithm
81             b = torch.zeros(batch_size, self.num_capsule, x.size(2)).to(x.device)
82             for i in range(self.routings):
83                 c = torch.softmax(b, dim=1) # Coupling coefficients
84                 s = (c.unsqueeze(-1) * u_hat).sum(dim=2) # Weighted sum
85                 v = self.squash(s) # Squashed output
86
87                 if i < self.routings - 1:
88                     # Update agreement values

```

```

89         agreement = (u_hat * v.unsqueeze(2)).sum(dim=-1)
90         b = b + agreement
91     return v
92
93 class CapsuleNet(nn.Module):
94     """Capsule Network architecture with primary and digit capsule layers."""
95     def __init__(self, input_dim):
96         super().__init__()
97         # Primary capsule layer
98         self.primary = nn.Sequential(
99             nn.Linear(input_dim, 256),
100            nn.ReLU(),
101            nn.Unflatten(1, (32, 8)) # Reshape to (batch, 32 capsules, 8 dim)
102        )
103
104        # Digit capsule layer (final classification layer)
105        self.digitcaps = CapsuleLayer(
106            num_capsule=2,          # Number of output classes
107            dim_capsule=16,         # Dimension of output capsules
108            input_num_capsule=32,   # Number of input capsules
109            input_dim_capsule=8,    # Dimension of input capsules
110            routings=3
111        )
112
113    def forward(self, x):
114        x = self.primary(x)
115        x = self.digitcaps(x)
116        # Calculate capsule magnitudes as class probabilities
117        x = torch.sqrt((x ** 2).sum(dim=-1))
118        return x
119
120 class MarginLoss(nn.Module):
121     """Margin loss for capsule network classification."""
122     def __init__(self):
123         super().__init__()
124
125     def forward(self, y_pred, y_true):
126         y_true = torch.nn.functional.one_hot(y_true, num_classes=2).float()
127         # Margin loss calculation
128         loss = y_true * torch.clamp(0.9 - y_pred, min=0) ** 2 + \
129             0.5 * (1 - y_true) * torch.clamp(y_pred - 0.1, min=0) ** 2
130         return loss.mean()
131
132     # Training configuration
133 BATCH_SIZE = 32
134 EPOCHS = 100
135 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
136
137     # Initialize metrics storage
138 accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
139 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
140
141     # 10-fold cross-validation loop

```

```
142 for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
143     X_train, X_val = X[train_idx], X[val_idx]
144     y_train, y_val = y[train_idx], y[val_idx]
145
146     # Create data loaders
147     train_loader = DataLoader(
148         TextDataset(X_train, y_train),
149         batch_size=BATCH_SIZE,
150         shuffle=True
151     )
152     val_loader = DataLoader(
153         TextDataset(X_val, y_val),
154         batch_size=BATCH_SIZE
155     )
156
157     # Initialize model and optimizer
158     model = CapsuleNet(X_train.shape[1]).to(device)
159     criterion = MarginLoss()
160     optimizer = optim.Adam(model.parameters(), lr=1e-3)
161
162     # Early stopping setup
163     best_val_loss = float('inf')
164     patience, counter = 10, 0
165
166     # Training loop
167     for epoch in range(EPOCHS):
168         model.train()
169         train_loss = 0
170         for inputs, labels in train_loader:
171             inputs, labels = inputs.to(device), labels.to(device)
172
173             optimizer.zero_grad()
174             outputs = model(inputs)
175             loss = criterion(outputs, labels)
176             loss.backward()
177             optimizer.step()
178
179         train_loss += loss.item()
180
181     # Validation phase
182     model.eval()
183     val_loss = 0
184     all_preds, all_labels = [], []
185     with torch.no_grad():
186         for inputs, labels in val_loader:
187             inputs, labels = inputs.to(device), labels.to(device)
188             outputs = model(inputs)
189             loss = criterion(outputs, labels)
190             val_loss += loss.item()
191
192             all_preds.append(outputs.cpu())
193             all_labels.append(labels.cpu())
194
```

```

195     val_loss /= len(val_loader)
196     all_preds = torch.cat(all_preds)
197     all_labels = torch.cat(all_labels)
198
199     # Early stopping check
200     if val_loss < best_val_loss:
201         best_val_loss = val_loss
202         counter = 0
203     else:
204         counter += 1
205         if counter >= patience:
206             break
207
208     # Final evaluation
209     model.eval()
210     y_pred, y_true = [], []
211     with torch.no_grad():
212         for inputs, labels in val_loader:
213             inputs = inputs.to(device)
214             outputs = model(inputs)
215             y_pred.append(outputs.cpu().numpy())
216             y_true.append(labels.numpy())
217
218     y_pred = np.concatenate(y_pred)
219     y_true = np.concatenate(y_true)
220     y_pred_labels = np.argmax(y_pred, axis=1)
221
222     # Calculate metrics
223     accuracies.append(accuracy_score(y_true, y_pred_labels))
224     precisions.append(precision_score(y_true, y_pred_labels, average='binary'))
225     recalls.append(recall_score(y_true, y_pred_labels, average='binary'))
226     f1_scores.append(f1_score(y_true, y_pred_labels, average='binary'))
227     aucs.append(roc_auc_score(y_true, y_pred[:, 1]))
228
229     # Print cross-validation results
230     print(f"Average Accuracy: {np.mean(accuracies):.4f}±{np.std(accuracies):.4f}")
231     print(f"Average Precision: {np.mean(precisions):.4f}±{np.std(precisions):.4f}")
232     print(f"Average Recall: {np.mean(recalls):.4f}±{np.std(recalls):.4f}")
233     print(f"Average F1-score: {np.mean(f1_scores):.4f}±{np.std(f1_scores):.4f}")
234     print(f"Average AUC-ROC: {np.mean(aucs):.4f}±{np.std(aucs):.4f}")

```

A. 4. 3 BertGCN

```

1 import pandas as pd
2 import numpy as np
3 import re
4 import jieba
5 import random
6 import math
7 import torch
8 import torch.nn as nn
9 import torch.nn.functional as F
10 from torch_geometric.data import Data

```

```
11 from torch_geometric.nn import GCNConv
12 from sklearn.model_selection import StratifiedKFold
13 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
14 from sklearn.feature_extraction.text import TfidfVectorizer
15 from collections import defaultdict
16 from transformers import AutoTokenizer, AutoModel
17
18 # Set random seeds for reproducibility
19 torch.manual_seed(42)
20 np.random.seed(42)
21 random.seed(42)
22
23 # -----
24 # Data Preprocessing
25 # -----
26
27 def remove_non_chinese(text):
28     """Remove non-Chinese characters from text"""
29     return re.sub(r'[\u4e00-\u9fa5]', '', text)
30
31 def tokenize(text, stop_words, mode='accurate'):
32     """Tokenize Chinese text using jieba with specified mode"""
33     if not isinstance(text, str):
34         return ""
35     text = text.strip()
36     if not text:
37         return ""
38
39     # Choose segmentation mode
40     if mode == 'full':
41         words = jieba.lcut(text, cut_all=True)
42     elif mode == 'search':
43         words = jieba.lcut_for_search(text)
44     else: # default accurate mode
45         words = jieba.lcut(text)
46
47     return " ".join([w for w in words if w not in stop_words])
48
49 # Load dataset
50 data = pd.read_excel("./Dataset.xlsx")
51 stopwords = [] # Placeholder for stopwords (empty in this example)
52
53 # Preprocess text
54 data['processed_text'] = data['text'].apply(
55     lambda x: tokenize(remove_non_chinese(str(x)), stopwords)
56 )
57
58 # -----
59 # Feature Engineering
60 # -----
61
62 # Build TF-IDF matrix
63 VOCAB_SIZE = 5000
```

```

64     vectorizer = TfidfVectorizer(
65         max_features=VOCAB_SIZE-1,
66         tokenizer=lambda x: x.split(),
67         lowercase=False
68     )
69     tfidf_matrix = vectorizer.fit_transform(data['processed_text'])
70     vocab = vectorizer.get_feature_names_out()
71
72     # -----
73     # Graph Construction
74     # -----
75
76     num_docs = len(data) # Document nodes
77     num_words = len(vocab) # Word nodes
78     word_node_ids = {word: num_docs+i for i, word in enumerate(vocab)} # Offset node IDs
79
80     # Document-Word edges (TF-IDF weights)
81     rows, cols = tfidf_matrix.nonzero()
82     doc_word_edges = [[row, word_node_ids[vocab[col]]] for row, col in zip(rows, cols)]
83     edge_weights = tfidf_matrix.data.tolist()
84
85     # Word-Word edges (PMI weights)
86     window_size = 20
87     word_doc_map = {word: set() for word in vocab}
88
89     # Build word-document mapping
90     for doc_id, doc in enumerate(data['processed_text']):
91         for word in doc.split():
92             if word in word_doc_map:
93                 word_doc_map[word].add(doc_id)
94
95     # Calculate co-occurrence statistics
96     cooccurrence = defaultdict(lambda: defaultdict(int))
97     for doc in data['processed_text']:
98         words = [w for w in doc.split() if w in vocab]
99         for i in range(len(words)):
100             context = words[max(0, i - window_size):i + window_size]
101             for neighbor in context:
102                 if neighbor != words[i]:
103                     cooccurrence[words[i]][neighbor] += 1
104
105     # Compute PMI scores
106     total_docs = len(data)
107     pmi_edges = []
108     for word1 in cooccurrence:
109         for word2 in cooccurrence[word1]:
110             p_word1 = len(word_doc_map[word1]) / total_docs
111             p_word2 = len(word_doc_map[word2]) / total_docs
112             p_cooccur = cooccurrence[word1][word2] / total_docs
113
114             # PMI calculation
115             pmi = math.log(p_cooccur / (p_word1 * p_word2)) if p_cooccur > 0 else 0
116             if pmi > 0:

```

```
117     src = word_node_ids[word1]
118     dst = word_node_ids[word2]
119     pmi_edges.extend([[src, dst], [dst, src]]) # Undirected edges
120
121 # Merge all edges
122 edge_index = doc_word_edges + pmi_edges
123 edge_index = torch.tensor(edge_index, dtype=torch.long).t().contiguous()
124 edge_attr = torch.tensor(edge_weights + [1.0]*len(pmi_edges), dtype=torch.float)
125
126 # -----
127 # Node Features
128 # -----
129
130 # Load BERT model for document embeddings
131 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
132 bert_model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
133 bert_model.eval() # Freeze BERT parameters
134
135 # Generate document embeddings using BERT
136 doc_embeddings = []
137 for text in data['text']:
138     inputs = tokenizer(
139         text,
140         return_tensors='pt',
141         padding=True,
142         truncation=True,
143         max_length=512
144     )
145     with torch.no_grad():
146         outputs = bert_model(**inputs)
147         cls_embedding = outputs.last_hidden_state[:, 0, :].squeeze().numpy()
148         doc_embeddings.append(cls_embedding)
149
150 doc_features = torch.tensor(np.array(doc_embeddings), dtype=torch.float)
151
152 # Initialize word node features randomly
153 word_features = torch.randn(num_words, 768) # Match BERT dimension
154
155 # Create full feature matrix
156 node_features = torch.cat([doc_features, word_features], dim=0)
157
158 # Create labels (only document nodes have labels)
159 labels = torch.full((num_docs + num_words,), -1, dtype=torch.long)
160 labels[:num_docs] = torch.tensor(data['class'].values, dtype=torch.long)
161
162 # Build PyG graph data object
163 graph = Data(
164     x=node_features,
165     edge_index=edge_index,
166     edge_attr=edge_attr,
167     y=labels
168 )
169
```

```
170 # -----
171 # Model Definition
172 # -----
173
174 class TextGCN(nn.Module):
175     """Graph Convolution Network for Text Classification"""
176     def __init__(self, hidden_dim, num_classes):
177         super().__init__()
178         self.conv1 = GCNConv(768, hidden_dim) # Input dim matches BERT
179         self.conv2 = GCNConv(hidden_dim, num_classes)
180         self.dropout = nn.Dropout(0.5)
181
182     def forward(self, data):
183         x = data.x # Use precomputed features
184         x = self.conv1(x, data.edge_index, data.edge_attr)
185         x = F.relu(x)
186         x = self.dropout(x)
187         x = self.conv2(x, data.edge_index, data.edge_attr)
188         return F.log_softmax(x, dim=1)
189
190 # -----
191 # Training & Evaluation
192 # -----
193
194 # 10-fold cross validation
195 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
196 metrics = []
197
198 for fold, (train_idx, test_idx) in enumerate(skf.split(np.zeros(num_docs), data['class'])):
199     print(f"\n==== Fold {fold+1}/10 ===")
200
201     # Create masks
202     train_mask = torch.cat([
203         torch.tensor([i in train_idx for i in range(num_docs)]),
204         torch.zeros(num_words, dtype=torch.bool)
205     ])
206
207     test_mask = torch.cat([
208         torch.tensor([i in test_idx for i in range(num_docs)]),
209         torch.zeros(num_words, dtype=torch.bool)
210     ])
211
212     # Initialize model and optimizer
213     model = TextGCN(hidden_dim=128, num_classes=2)
214     optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
215
216     # Early stopping parameters
217     best_loss = float('inf')
218     patience_counter = 0
219     PATIENCE = 10
220
221     # Training loop
222     for epoch in range(100):
```

```
223     model.train()
224     optimizer.zero_grad()
225     outputs = model(graph)
226
227     # Calculate training loss
228     loss = F.nll_loss(outputs[train_mask], graph.y[train_mask])
229     loss.backward()
230     optimizer.step()
231
232     # Validation (using test set in cross-validation)
233     with torch.no_grad():
234         test_outputs = model(graph)
235         test_loss = F.nll_loss(test_outputs[test_mask], graph.y[test_mask])
236
237     # Early stopping check
238     if test_loss < best_loss:
239         best_loss = test_loss
240         patience_counter = 0
241         best_state = model.state_dict().copy()
242     else:
243         patience_counter += 1
244         if patience_counter >= PATIENCE:
245             break
246
247     # Load best model weights
248     model.load_state_dict(best_state)
249
250     # Final evaluation
251     model.eval()
252     with torch.no_grad():
253         logits = model(graph)
254         preds = logits[test_mask].argmax(dim=1)
255         y_true = graph.y[test_mask].numpy()
256         y_pred = preds.numpy()
257
258         metrics.append({
259             'accuracy': accuracy_score(y_true, y_pred),
260             'precision': precision_score(y_true, y_pred),
261             'recall': recall_score(y_true, y_pred),
262             'f1': f1_score(y_true, y_pred),
263             'auc': roc_auc_score(y_true, y_pred)
264         })
265
266     # -----
267     # Results Reporting
268     # -----
269
270     print(f"Average Accuracy: {np.mean([m['accuracy'] for m in metrics]):.4f} ±{np.std([m['accuracy'] for m in metrics]):.4f}")
271     print(f"Average Precision: {np.mean([m['precision'] for m in metrics]):.4f}±{np.std([m['precision'] for m in metrics]):.4f}")
272     print(f"Average Recall: {np.mean([m['recall'] for m in metrics]):.4f}±{np.std([m['recall'] for m in metrics]):.4f}")
273     print(f"Average F1: {np.mean([m['f1'] for m in metrics]):.4f}±{np.std([m['f1'] for m in metrics]):.4f}")
```

```
274 print(f"Average AUC-ROC: {np.mean([m['auc'] for m in metrics]):.4f}±{np.std([m['auc'] for m in metrics]):.4f}")
```

A. 4. 4 HBGA

```

1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 from torch_geometric.nn import GATConv
5 from transformers import AutoTokenizer, AutoModel
6 import numpy as np
7 import pandas as pd
8 from sklearn.model_selection import StratifiedKFold
9 from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
10 from torch.utils.data import Dataset, DataLoader, Subset
11 from tqdm import tqdm
12
13 # Set random seeds for reproducibility
14 torch.manual_seed(42)
15 np.random.seed(42)
16
17 # 1. ERNIE Context Encoder
18 class ErnieEncoder(nn.Module):
19     """ERNIE-based text encoder with frozen weights for contextual embeddings"""
20     def __init__(self):
21         super().__init__()
22         self.tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
23         self.ernie = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
24
25     # Freeze ERNIE parameters
26     for param in self.ernie.parameters():
27         param.requires_grad = False
28
29     def forward(self, texts):
30         """Input: list of texts | Output: contextual embeddings [batch, seq_len, 768]"""
31         with torch.no_grad(): # Disable gradient calculation
32             inputs = self.tokenizer(texts, return_tensors='pt', padding=True, truncation=True, max_length=150)
33             outputs = self.ernie(**inputs)
34             return outputs.last_hidden_state
35
36 # 2. Bi-LSTM Global Feature Extractor
37 class BiLSTMGlobal(nn.Module):
38     """Bi-directional LSTM for capturing global semantic features"""
39     def __init__(self, input_dim=768, hidden_dim=256):
40         super().__init__()
41         self.lstm = nn.LSTM(input_dim, hidden_dim, bidirectional=True, batch_first=True)
42
43     def forward(self, x):
44         outputs, (h_n, _) = self.lstm(x)
45         # Concatenate last hidden states from both directions
46         return torch.cat((h_n[-2], h_n[-1]), dim=1)
47
48 # 3. Capsule Network for Local Feature Extraction
49 class CapsuleLayer(nn.Module):

```

```

50     """Capsule network with dynamic routing to capture local patterns"""
51     def __init__(self, num_capsules=8, in_channels=768, out_channels=32, routings=3):
52         super().__init__()
53         self.num_capsules = num_capsules
54         self.routings = routings
55         self.capsules = nn.ModuleList([
56             nn.Conv1d(in_channels, out_channels, kernel_size=3, padding=1)
57             for _ in range(num_capsules)])
58
59     def squash(self, tensor, dim=-1):
60         """Non-linear 'squashing' function for capsule outputs"""
61         squared_norm = (tensor ** 2).sum(dim=dim, keepdim=True)
62         scale = squared_norm / (1 + squared_norm)
63         return scale * tensor / torch.sqrt(squared_norm + 1e-8)
64
65     def forward(self, x):
66         batch_size, seq_len, _ = x.size()
67         x = x.permute(0, 2, 1) # [batch, channels, seq]
68
69         # Generate candidate capsules
70         u_hat = [caps(x).permute(0, 2, 1) for caps in self.capsules]
71         u_hat = torch.stack(u_hat, dim=1) # [batch, num_caps, seq, out_channels]
72
73         # Dynamic routing mechanism
74         b = torch.zeros(batch_size, self.num_capsules, seq_len).to(x.device)
75         for i in range(self.routings):
76             c = F.softmax(b, dim=1)
77             s = (c.unsqueeze(-1) * u_hat).sum(dim=2)
78             v = self.squash(s)
79
80             if i < self.routings - 1:
81                 agreement = (u_hat * v.unsqueeze(2)).sum(dim=-1)
82                 b = b + agreement
83
84         return v # [batch, num_caps, out_channels]
85
86     # 4. GAT for Label Correlation Modeling
87     class LabelGAT(nn.Module):
88         """Graph Attention Network to model label relationships"""
89         def __init__(self, num_labels, emb_dim=128):
90             super().__init__()
91             self.label_emb = nn.Embedding(num_labels, emb_dim)
92             self.gat = GATConv(emb_dim, emb_dim, heads=2, concat=False)
93             self.edge_index = self.build_label_graph(num_labels) # Example: fully connected
94
95         def build_label_graph(self, num_labels):
96             """Create label graph structure (default: fully connected)"""
97             return torch.combinations(torch.arange(num_labels), 2).t().contiguous()
98
99         def forward(self):
100            x = self.label_emb.weight
101            return self.gat(x, self.edge_index) # [num_labels, emb_dim]
102

```

```
103 # 5. Integrated Model Architecture
104 class ErnieBiLSTMCapsGAT(nn.Module):
105     """Main model combining all components for hierarchical feature fusion"""
106     def __init__(self, num_labels):
107         super().__init__()
108         self.ernie = ErnieEncoder()
109         self.bilstm = BiLSTMGlobal()
110         self.capsule = CapsuleLayer()
111         self.gat = LabelGAT(num_labels)
112
113     # Feature projection layers
114     self.global_proj = nn.Linear(512, 128) # BiLSTM output: 512
115     self.local_proj = nn.Linear(8*32, 128) # Capsule output: 8*32
116     self.final_fc = nn.Linear(128*2 + 128, num_labels) # Global+Local+Label features
117
118     def forward(self, texts):
119         # Contextual encoding
120         contextual = self.ernie(texts) # [batch, seq, 768]
121
122         # Global feature extraction
123         global_feat = self.bilstm(contextual) # [batch, 512]
124         global_feat = self.global_proj(global_feat) # [batch, 128]
125
126         # Local feature extraction
127         caps_out = self.capsule(contextual) # [batch, 8, 32]
128         local_feat = caps_out.view(caps_out.size(0), -1) # [batch, 256]
129         local_feat = self.local_proj(local_feat) # [batch, 128]
130
131         # Label correlation features
132         label_feat = self.gat() # [num_labels, 128]
133         label_feat = label_feat.mean(dim=0).unsqueeze(0) # [1, 128]
134         label_feat = label_feat.expand(global_feat.size(0), -1) # [batch, 128]
135
136         # Feature fusion
137         combined = torch.cat([global_feat, local_feat, label_feat], dim=1)
138         return F.log_softmax(self.final_fc(combined), dim=1)
139
140     # Dataset Class
141     class TextDataset(Dataset):
142         """Custom dataset for text classification"""
143         def __init__(self, texts, labels):
144             self.texts = texts
145             self.labels = labels
146
147         def __len__(self):
148             return len(self.texts)
149
150         def __getitem__(self, idx):
151             return self.texts[idx], self.labels[idx]
152
153     # Training Configuration
154     BATCH_SIZE = 32
155     MAX_EPOCHS = 100
```

```
156 LEARNING_RATE = 1e-3
157 PATIENCE = 10
158 N_SPLITS = 10 # 10-fold cross validation
159 RANDOM_STATE = 42
160
161 # Load dataset
162 data = pd.read_excel("./Dataset.xlsx")
163 texts = data['text'].tolist()
164 labels = data['class'].values
165
166 # Create full dataset
167 full_dataset = TextDataset(texts, labels)
168
169 # Initialize stratified K-fold
170 skf = StratifiedKFold(n_splits=N_SPLITS, shuffle=True, random_state=RANDOM_STATE)
171
172 def evaluate(model, loader, device, criterion):
173     """Evaluation function for model performance metrics"""
174     model.eval()
175     total_loss = 0
176     all_preds = []
177     all_labels = []
178
179     with torch.no_grad():
180         for batch in tqdm(loader, desc="Evaluating"):
181             texts, labels = batch
182             labels = torch.tensor(labels).to(device)
183
184             outputs = model(texts)
185             loss = criterion(outputs, labels)
186
187             total_loss += loss.item()
188             all_preds.extend(outputs.argmax(dim=1).cpu().numpy())
189             all_labels.extend(labels.cpu().numpy())
190
191     metrics = {
192         'loss': total_loss / len(loader),
193         'accuracy': accuracy_score(all_labels, all_preds),
194         'precision': precision_score(all_labels, all_preds),
195         'recall': recall_score(all_labels, all_preds),
196         'f1': f1_score(all_labels, all_preds),
197         'auc': roc_auc_score(all_labels, all_preds)
198     }
199     return metrics
200
201 # Store cross-validation metrics
202 fold_metrics = {
203     'accuracy': [],
204     'precision': [],
205     'recall': [],
206     'f1': [],
207     'auc': []
208 }
```

```
209
210 # Cross-validation loop
211 for fold, (train_idx, val_idx) in enumerate(skf.split(texts, labels)):
212     print(f"\n==== Fold {fold+1}/{N_SPLITS} ====")
213
214 # Create data loaders
215 train_loader = DataLoader(
216     Subset(full_dataset, train_idx),
217     batch_size=BATCH_SIZE,
218     shuffle=True,
219     collate_fn=lambda x: zip(*x)
220 )
221 val_loader = DataLoader(
222     Subset(full_dataset, val_idx),
223     batch_size=BATCH_SIZE,
224     collate_fn=lambda x: zip(*x)
225 )
226
227 # Initialize model and optimizer
228 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
229 model = ErnieBiLSTMCapsGAT(num_labels=2).to(device)
230 optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
231 criterion = nn.NLLLoss()
232
233 # Training variables
234 best_val_loss = float('inf')
235 no_improve = 0
236 history = {'train_loss': [], 'val_loss': [], 'val_f1': []}
237
238 # Training loop
239 for epoch in range(MAX_EPOCHS):
240     print(f"\nEpoch {epoch+1}/{MAX_EPOCHS}")
241
242     # Training phase
243     model.train()
244     total_loss = 0
245     for batch in tqdm(train_loader, desc="Training"):
246         texts, labels = batch
247         labels = torch.tensor(labels).to(device)
248
249         optimizer.zero_grad()
250         outputs = model(texts)
251         loss = criterion(outputs, labels)
252
253         loss.backward()
254         torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
255         optimizer.step()
256
257         total_loss += loss.item()
258     train_loss = total_loss / len(train_loader)
259     history['train_loss'].append(train_loss)
260
261     # Validation phase
```

```
262     model.eval()
263     total_loss = 0
264     all_preds = []
265     all_labels = []
266     with torch.no_grad():
267         for batch in tqdm(val_loader, desc="Evaluating"):
268             texts, labels = batch
269             labels = torch.tensor(labels).to(device)
270
271             outputs = model(texts)
272             loss = criterion(outputs, labels)
273
274             total_loss += loss.item()
275             all_preds.extend(outputs.argmax(dim=1).cpu().numpy())
276             all_labels.extend(labels.cpu().numpy())
277
278             val_loss = total_loss / len(val_loader)
279             val_acc = accuracy_score(all_labels, all_preds)
280             val_f1 = f1_score(all_labels, all_preds)
281             val_auc = roc_auc_score(all_labels, all_preds)
282
283             history['val_loss'].append(val_loss)
284             history['val_f1'].append(val_f1)
285
286             # Early stopping logic
287             if val_loss < best_val_loss:
288                 best_val_loss = val_loss
289                 no_improve = 0
290             else:
291                 no_improve += 1
292             if no_improve >= PATIENCE:
293                 print(f"Early stopping at epoch {epoch+1}")
294                 break
295
296             # Print metrics
297             print(f"Train Loss: {train_loss:.4f} | Val Loss: {val_loss:.4f}")
298             print(f"Val Acc: {val_acc:.4f} | Val F1: {val_f1:.4f} | Val AUC: {val_auc:.4f}")
299
300             # Final evaluation
301             final_metrics = evaluate(model, val_loader, device, criterion)
302
303             # Store fold metrics
304             for metric in fold_metrics:
305                 fold_metrics[metric].append(final_metrics[metric])
306
307             # Output cross-validation results
308             print("\n==== Cross Validation Results ====")
309             print(f"Average Accuracy: {np.mean(fold_metrics['accuracy']):.4f}±{np.std(fold_metrics['accuracy']):.4f}")
310             print(f"Average Precision: {np.mean(fold_metrics['precision']):.4f}±{np.std(fold_metrics['precision']):.4f}")
311             print(f"Average Recall: {np.mean(fold_metrics['recall']):.4f}±{np.std(fold_metrics['recall']):.4f}")
312             print(f"Average F1-Score: {np.mean(fold_metrics['f1']):.4f}±{np.std(fold_metrics['f1']):.4f}")
313             print(f"Average AUC-ROC: {np.mean(fold_metrics['auc']):.4f}±{np.std(fold_metrics['auc']):.4f}")
```

A. 4. 5 A-Hybrid

```
1 import random
2 import pandas as pd
3 import numpy as np
4 import tensorflow as tf
5 tf.config.set_visible_devices([], 'GPU') # Disable GPU usage for TensorFlow
6 import os
7 os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com/' # Set HuggingFace mirror endpoint
8 from keras import regularizers
9 from keras.models import Model
10 from keras.layers import Input, Embedding, Conv1D, GlobalMaxPooling1D, Dense, Bidirectional, Dropout, LSTM
11 from keras.callbacks import EarlyStopping
12 from sklearn.model_selection import StratifiedKFold
13 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
14 import torch
15 from transformers import AutoTokenizer, AutoModel
16
17 # Set random seeds for reproducibility
18 tf.random.set_seed(42)
19 random.seed(42)
20 np.random.seed(42)
21
22 # Read dataset from Excel file
23 data = pd.read_excel("./Dataset.xlsx")
24
25 # Load pre-trained BERT model and tokenizer
26 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
27 model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
28
29 def extract_bert_embeddings(text, tokenizer, model, max_length=100):
30     """
31     Extract BERT embeddings for input text.
32
33     Args:
34         text (str): Input text
35         tokenizer: BERT tokenizer
36         model: BERT model
37         max_length (int): Maximum sequence length
38
39     Returns:
40         np.array: Sentence embeddings (shape: [max_length, hidden_size])
41     """
42     inputs = tokenizer(text, return_tensors='pt',
43                       max_length=max_length,
44                       padding='max_length',
45                       truncation=True)
46
47     with torch.no_grad():
48         outputs = model(**inputs)
49
50     embeddings = outputs.last_hidden_state
51     return embeddings.squeeze().cpu().numpy()
52
```

```
53 # Generate BERT embeddings for all texts
54 X = np.array([extract_bert_embeddings(text, tokenizer, model, max_length=150) for text in data['text']])
55 y = data['class'].values # Target labels
56
57 class LuongAttention(tf.keras.Model):
58     """Implementation of Luong Attention Mechanism"""
59     def __init__(self, rnn_size, attention_func):
60         super(LuongAttention, self).__init__()
61         self.attention_func = attention_func
62
63     if attention_func not in ['dot', 'general', 'concat']:
64         raise ValueError('Invalid attention function. Choose from dot/general(concat).')
65
66     # Initialize parameters based on attention type
67     if attention_func == 'general':
68         self.wa = tf.keras.layers.Dense(rnn_size)
69     elif attention_func == 'concat':
70         self.wa = tf.keras.layers.Dense(rnn_size, activation='tanh')
71         self.va = tf.keras.layers.Dense(1)
72
73     def call(self, decoder_output, encoder_output):
74         """Calculate context vector and alignment"""
75         # Compute attention scores
76         if self.attention_func == 'dot':
77             score = tf.matmul(decoder_output, encoder_output, transpose_b=True)
78         elif self.attention_func == 'general':
79             score = tf.matmul(decoder_output, self.wa(encoder_output), transpose_b=True)
80         elif self.attention_func == 'concat':
81             decoder_output = tf.tile(decoder_output, [1, encoder_output.shape[1], 1])
82             score = self.va(self.wa(tf.concat((decoder_output, encoder_output), axis=-1)))
83             score = tf.transpose(score, [0, 2, 1])
84
85         # Compute attention weights
86         alignment = tf.nn.softmax(score, axis=2)
87         context = tf.matmul(alignment, encoder_output)
88         return context, alignment
89
90     # Initialize attention layer
91     Attention = LuongAttention(20, 'dot')
92
93     def hybrid_model(input_shape, kernel_size=3):
94         """Build hybrid LSTM-CNN-Attention model"""
95         inputs = Input(shape=input_shape)
96
97         # Bidirectional LSTM layer
98         x = Bidirectional(LSTM(100, return_sequences=True,
99                             activation='tanh',
100                            recurrent_activation='sigmoid',
101                            use_bias=True))(inputs)
102        # Stacked LSTM layer
103        x = LSTM(100, return_sequences=True,
104                 activation='tanh',
105                 recurrent_activation='sigmoid',
```

```
106     use_bias=True)(x)
107
108 # 1D Convolutional layer
109 x = Conv1D(50, kernel_size, activation='relu')(x)
110
111 # Attention layer
112 x = Attention(x, x)[0]
113
114 # Regularized dense layer
115 x = Dense(50, activation="relu",
116             kernel_regularizer=regularizers.L1L2(l1=1e-5, l2=1e-4))(x)
117
118 # Global max pooling and classifier head
119 x = GlobalMaxPooling1D()(x)
120 x = Dense(250, activation="relu")(x)
121 x = Dropout(0.5)(x)
122 x = Dense(50, activation="relu")(x)
123 x = Dropout(0.5)(x)
124 outputs = Dense(1, activation="sigmoid")(x)
125
126 model = Model(inputs=inputs, outputs=outputs)
127 model.compile(loss='binary_crossentropy',
128                 optimizer='adam',
129                 metrics=['accuracy'])
130 return model
131
132 # Training parameters
133 BATCH_SIZE = 32
134 EPOCHS = 100
135
136 # Initialize metrics storage
137 accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
138 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
139
140 # Cross-validation loop
141 for train_idx, val_idx in skf.split(X, y):
142     X_train, X_val = X[train_idx], X[val_idx]
143     y_train, y_val = y[train_idx], y[val_idx]
144
145 # Build and train model
146 model = hybrid_model((X_train.shape[1], X_train.shape[2]))
147 model.fit(X_train, y_train,
148             batch_size=BATCH_SIZE,
149             epochs=EPOCHS,
150             validation_data=(X_val, y_val),
151             verbose=0,
152             callbacks=[EarlyStopping(patience=10, restore_best_weights=True)])
153
154 # Evaluate model
155 y_pred = (model.predict(X_val) > 0.5).astype(int)
156 accuracies.append(accuracy_score(y_val, y_pred))
157 precisions.append(precision_score(y_val, y_pred))
158 recalls.append(recall_score(y_val, y_pred))
```

```

159     f1_scores.append(f1_score(y_val, y_pred))
160     aucs.append(roc_auc_score(y_val, y_pred))
161
162     # Print cross-validation results
163     print(f"Average cross-validation accuracy: {np.mean(accuracies):.4f}±{np.std(accuracies):.4f}")
164     print(f"Average cross-validation precision: {np.mean(precisions):.4f}±{np.std(precisions):.4f}")
165     print(f"Average cross-validation recall: {np.mean(recalls):.4f}±{np.std(recalls):.4f}")
166     print(f"Average cross-validation F1: {np.mean(f1_scores):.4f}±{np.std(f1_scores):.4f}")
167     print(f"Average cross-validation AUC-ROC: {np.mean(aucs):.4f}±{np.std(aucs):.4f}")

```

A. 4. 6 M-BERT

```

1 import os
2 os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com/' # Set HuggingFace mirror for China users
3 import pandas as pd
4 import numpy as np
5 import torch
6 import torch.nn as nn
7 import random
8 from torch.utils.data import Dataset, DataLoader
9 from transformers import AutoTokenizer, AutoModel
10 from sklearn.model_selection import StratifiedKFold
11 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
12 from sklearn.decomposition import LatentDirichletAllocation
13 from sklearn.feature_extraction.text import CountVectorizer
14
15 # Set random seeds for reproducibility
16 SEED = 42
17 random.seed(SEED)
18 torch.manual_seed(SEED)
19 np.random.seed(SEED)
20 if torch.cuda.is_available():
21     torch.cuda.manual_seed_all(SEED)
22
23 # Load dataset
24 dataset = pd.read_excel("./Dataset.xlsx")
25
26 # Hardware configuration
27 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
28
29 # Hyperparameters
30 N_SPLITS = 10      # Number of cross-validation folds
31 MAX_LENGTH = 150    # Maximum sequence length for tokenizer
32 BATCH_SIZE = 32      # Training batch size
33 EPOCHS = 100       # Maximum training epochs
34 PATIENCE = 10       # Early stopping patience
35 LEARNING_RATE = 1e-3   # Learning rate
36 N_COMPONENTS = 10      # Number of LDA topics
37
38 # Custom Dataset Class for multimodal data (text + LDA features)
39 class FusionDataset(Dataset):
40     """Dataset class combining text inputs and LDA topic features"""
41     def __init__(self, texts, lda_features, labels, tokenizer, max_length):

```

```
42     self.texts = texts
43     self.lda_features = lda_features
44     self.labels = labels
45     self.tokenizer = tokenizer
46     self.max_length = max_length
47
48     def __len__(self):
49         return len(self.texts)
50
51     def __getitem__(self, idx):
52         text = str(self.texts[idx])
53         label = self.labels[idx]
54         lda_feature = self.lda_features[idx]
55
56         # Tokenize text input
57         encoding = self.tokenizer.encode_plus(
58             text,
59             add_special_tokens=True,
60             max_length=self.max_length,
61             padding='max_length',
62             truncation=True,
63             return_attention_mask=True,
64             return_tensors='pt'
65         )
66
67         return {
68             'input_ids': encoding['input_ids'].flatten(),
69             'attention_mask': encoding['attention_mask'].flatten(),
70             'lda_features': torch.tensor(lda_feature, dtype=torch.float32),
71             'labels': torch.tensor(label, dtype=torch.float)
72         }
73
74     # Multimodal Model Architecture
75     class LDWithErnieModel(nn.Module):
76         """ERNIE model with LDA feature integration through attention mechanism"""
77         def __init__(self, num_classes, num_topics):
78             super().__init__()
79             # Initialize frozen ERNIE model
80             self.ernie = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
81             for param in self.ernie.parameters():
82                 param.requires_grad = False
83
84             # Feature transformation layers
85             self.hidden_size = self.ernie.config.hidden_size
86             self.linear_ernie = nn.Linear(self.hidden_size, 128)
87             self.linear_lda = nn.Linear(num_topics, 128)
88
89             # Feature fusion layer
90             self.linear_splice = nn.Linear(256, self.hidden_size)
91             self.classifier = nn.Linear(self.hidden_size, num_classes)
92             self.sigmoid = nn.Sigmoid()
93
94         def forward(self, input_ids, attention_mask, lda_features):
```

```
95     # Initial ERNIE processing
96     outputs = self.ernie(input_ids=input_ids, attention_mask=attention_mask)
97     sequence_output = outputs.last_hidden_state # [batch, seq_len, hidden_size]
98
99     # Transform ERNIE features
100    attention_input = self.linear_ernie(sequence_output)
101
102    # Prepare LDA features for attention gate
103    repeated_lda = torch.stack([lda_features] * attention_input.size(1), dim=1).to(device)
104
105    # Attention-based feature fusion
106    AG = AttentionGate(attention_input.shape[1]).to(device)
107    attention_gate = AG(attention_input, repeated_lda)
108
109    # Feature refinement
110    shift_vector = self.linear_splice(attention_gate)
111    multi_shifting = sequence_output + shift_vector
112
113    # Final classification
114    outputs_new = self.ernie(inputs_embeds=multi_shifting, attention_mask=attention_mask)
115    cls_output = outputs_new.last_hidden_state[:, 0, :] # CLS token
116    logits = self.sigmoid(self.classifier(cls_output))
117
118    return logits
119
120    # Attention Mechanism Module
121    class AttentionGate(nn.Module):
122        """Custom attention mechanism for fusing LDA features with text embeddings"""
123        def __init__(self, seq_length):
124            super().__init__()
125            # Position-wise transformation layers
126            self.fc_layers = nn.ModuleList([nn.Linear(10, 128) for _ in range(seq_length)])
127
128        def forward(self, attention_input, features_output):
129            batch_size, seq_length, _ = attention_input.size()
130            transformed_features = []
131
132            # Apply position-specific transformations
133            for i in range(seq_length):
134                fc = self.fc_layers[i]
135                transformed = fc(features_output[:, i, :])
136                transformed_features.append(transformed)
137
138            # Concatenate transformed features
139            attention_gate = torch.stack(transformed_features, dim=1)
140            # Concatenate with original attention inputs
141            combined = torch.cat((attention_input, attention_gate), dim=2)
142
143            return combined
144
145    # Initialize stratified K-fold cross-validation
146    skf = StratifiedKFold(n_splits=N_SPLITS, shuffle=True, random_state=SEED)
```

```
148 # Metrics storage
149 fold_results = []
150 test_metrics = {
151     'accuracies': [],
152     'f1_scores': [],
153     'precisions': [],
154     'recalls': [],
155     'aucs': []
156 }
157
158 # Preprocess text data
159 texts = [text.replace("。", "").replace("，", "").replace("“", "“").replace("”", "”") for text in dataset['text']] # Basic Chinese text cleaning
160 vectorizer = CountVectorizer()
161 X_counts = vectorizer.fit_transform(texts)
162 lda = LatentDirichletAllocation(n_components=N_COMPONENTS, random_state=SEED)
163 lda_features = lda.fit_transform(X_counts)
164
165 # Main cross-validation loop
166 for fold, (train_idx, val_idx) in enumerate(skf.split(lda_features, dataset['class'])):
167     print(f"\n{'='*30} Fold {fold+1}/{N_SPLITS} {'='*30}")
168
169 # Data splitting
170 X_train, X_val = lda_features[train_idx], lda_features[val_idx]
171 y_train, y_val = dataset['class'].iloc[train_idx], dataset['class'].iloc[val_idx]
172
173 # Initialize tokenizer and data loaders
174 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
175 train_dataset = FusionDataset(
176     texts=[dataset['text'][i] for i in train_idx],
177     lda_features=X_train,
178     labels=y_train.values,
179     tokenizer=tokenizer,
180     max_length=MAX_LENGTH
181 )
182 val_dataset = FusionDataset(
183     texts=[dataset['text'][i] for i in val_idx],
184     lda_features=X_val,
185     labels=y_val.values,
186     tokenizer=tokenizer,
187     max_length=MAX_LENGTH
188 )
189
190 train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
191 val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE)
192
193 # Model initialization
194 model = LDAwithErnieModel(num_classes=1, num_topics=N_COMPONENTS).to(device)
195 optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
196 criterion = nn.BCELoss()
197
198 # Training tracking
199 best_metrics = {'acc': 0, 'f1': 0, 'recall': 0,
200                 'precision': 0, 'auc': 0, 'epoch': 0, 'loss': float('inf')}
```

```
201     no_improve = 0
202
203     # Training loop
204     for epoch in range(EPOCHS):
205         model.train()
206         total_loss = 0
207
208         # Batch training
209         for batch in train_loader:
210             optimizer.zero_grad()
211
212             inputs = {k: v.to(device) for k, v in batch.items() if k != 'labels'}
213             labels = batch['labels'].to(device)
214
215             outputs = model(**inputs).squeeze(-1)
216             loss = criterion(outputs, labels)
217
218             loss.backward()
219             optimizer.step()
220             total_loss += loss.item()
221
222         # Validation phase
223         model.eval()
224         val_preds, val_true, val_probs = [], [], []
225         val_loss = 0
226
227         with torch.no_grad():
228             for batch in val_loader:
229                 inputs = {k: v.to(device) for k, v in batch.items() if k != 'labels'}
230                 labels = batch['labels'].to(device)
231
232                 outputs = model(**inputs).squeeze(-1)
233                 loss = criterion(outputs, labels)
234                 val_loss += loss.item()
235
236                 preds = (outputs > 0.5).float().cpu().numpy()
237                 val_preds.extend(preds)
238                 val_true.extend(labels.cpu().numpy())
239
240         # Calculate metrics
241         val_acc = accuracy_score(val_true, val_preds)
242         val_pre = precision_score(val_true, val_preds)
243         val_rec = recall_score(val_true, val_preds)
244         val_f1 = f1_score(val_true, val_preds)
245
246         try:
247             val_auc = roc_auc_score(val_true, val_preds)
248         except ValueError:
249             val_auc = 0.0 # Handle cases with single class
250
251         # Early stopping check
252         if val_loss < best_metrics['loss']:
253             best_metrics.update(acc=val_acc, f1=val_f1, recall=val_rec,
254                                 precision=val_pre, auc=val_auc,
```

```

254             epoch=epoch, loss=val_loss)
255         no_improve = 0
256     else:
257         no_improve += 1
258
259     # Training progress report
260     if (epoch+1) % 5 == 0:
261         print(f"Epoch {epoch+1}/{EPOCHS}")
262         print(f"Train Loss: {total_loss/len(train_loader):.4f} | Val Loss: {val_loss/len(val_loader):.4f}")
263         print(f"Val Acc: {val_acc:.4f} | F1: {val_f1:.4f} | AUC: {val_auc:.4f}")
264
265     if no_improve >= PATIENCE:
266         print(f"Early stopping at epoch {epoch+1}")
267         break
268
269     # Store fold results
270     test_metrics['accuracies'].append(best_metrics['acc'])
271     test_metrics['f1_scores'].append(best_metrics['f1'])
272     test_metrics['precisions'].append(best_metrics['precision'])
273     test_metrics['recalls'].append(best_metrics['recall'])
274     test_metrics['aucs'].append(best_metrics['auc'])
275
276     fold_results.append({
277         'fold': fold+1,
278         'accuracy': best_metrics['acc'],
279         'precision': best_metrics['precision'],
280         'recall': best_metrics['recall'],
281         'f1_score': best_metrics['f1'],
282         'auc': best_metrics['auc'],
283         'best_epoch': best_metrics['epoch']
284     })
285
286     # Cleanup resources
287     del model
288     torch.cuda.empty_cache()
289
290     # Final performance report
291     print("\nFinal Cross-Validation Results:")
292     print(f"Average Accuracy: {np.mean(test_metrics['accuracies']):.4f}±{np.std(test_metrics['accuracies']):.4f}")
293     print(f"Average F1 Score: {np.mean(test_metrics['f1_scores']):.4f}±{np.std(test_metrics['f1_scores']):.4f}")
294     print(f"Average Precision: {np.mean(test_metrics['precisions']):.4f}±{np.std(test_metrics['precisions']):.4f}")
295     print(f"Average Recall: {np.mean(test_metrics['recalls']):.4f}±{np.std(test_metrics['recalls']):.4f}")
296     print(f"Average AUC: {np.mean(test_metrics['aucs']):.4f}±{np.std(test_metrics['aucs']):.4f}")
297
298     # Save results
299     pd.DataFrame(fold_results).to_csv("cv_results.csv", index=False)

```

A. 4. 7 CBA

```

1 import random
2 import pandas as pd
3 import numpy as np
4 import tensorflow as tf

```

```
5  tf.config.set_visible_devices([], 'GPU') # Disable GPU usage for TensorFlow
6  import os
7  os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com/' # Set HuggingFace mirror
8  from keras.models import Model
9  from keras.layers import Input, Embedding, Conv1D, Dense, Bidirectional, LSTM, SpatialDropout1D, Attention,
   MaxPooling1D
10 from keras.callbacks import EarlyStopping
11 from sklearn.model_selection import StratifiedKFold
12 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
13 import torch
14 from transformers import AutoTokenizer, AutoModel
15
16 # Set random seeds for reproducibility
17 tf.random.set_seed(42)
18 random.seed(42)
19 np.random.seed(42)
20
21 # Load dataset
22 data = pd.read_excel("./Dataset.xlsx")
23
24 # Load pretrained BERT model and tokenizer
25 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
26 bert_model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')
27
28 def extract_bert_embeddings(text, tokenizer, model, max_length=100):
29     """
30         Generate sentence embeddings using BERT model.
31
32     Args:
33         text (str): Input text
34         tokenizer: BERT tokenizer
35         model: BERT model
36         max_length (int): Maximum sequence length
37
38     Returns:
39         np.array: Sentence embeddings
40     """
41     inputs = tokenizer(text,
42                         return_tensors='pt',
43                         max_length=max_length,
44                         padding='max_length',
45                         truncation=True)
46
47     with torch.no_grad():
48         outputs = model(**inputs)
49
50     embeddings = outputs.last_hidden_state
51     return embeddings.squeeze().cpu().numpy()
52
53 # Generate BERT embeddings for all texts
54 X = np.array([extract_bert_embeddings(text, tokenizer, bert_model, max_length=150)
55               for text in data['text']])
56 y = data['class'].values # Target labels
```

```
57
58 def build_hybrid_model(input_shape, kernel_size=5):
59     """
60     Build hybrid CNN-BiLSTM-Attention model.
61
62     Args:
63         input_shape (tuple): Shape of input data
64         kernel_size (int): Size of convolutional kernels
65
66     Returns:
67         keras.Model: Compiled Keras model
68     """
69     inputs = Input(shape=input_shape)
70
71     # Feature extraction block
72     x = SpatialDropout1D(0.2)(inputs)
73     x = Conv1D(128, kernel_size, activation='relu', strides=1)(x)
74     x = MaxPooling1D(strides=1, pool_size=3)(x)
75     x = Conv1D(128, kernel_size, activation='relu', strides=1)(x)
76     x = MaxPooling1D(strides=1, pool_size=3)(x)
77     x = Conv1D(128, kernel_size, activation='relu', strides=1)(x)
78     x = MaxPooling1D(strides=1, pool_size=3)(x)
79
80     # Temporal processing block
81     x = Bidirectional(LSTM(256, return_sequences=False, activation='relu'))(x)
82
83     # Attention and classification block
84     x = Attention()([x, x])
85     x = Dense(96, activation="relu")(x)
86     outputs = Dense(1, activation="sigmoid")(x)
87
88     model = Model(inputs=inputs, outputs=outputs)
89     model.compile(loss='binary_crossentropy',
90                     optimizer='adam',
91                     metrics=['accuracy'])
92     return model
93
94     # Training parameters
95 BATCH_SIZE = 32
96 EPOCHS = 100
97
98     # Initialize metrics storage
99 accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
100
101    # 10-fold stratified cross-validation
102 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
103
104 for fold, (train_idx, val_idx) in enumerate(skf.split(X, y)):
105     print(f"\nProcessing fold {fold+1}/10")
106     X_train, X_val = X[train_idx], X[val_idx]
107     y_train, y_val = y[train_idx], y[val_idx]
108
109     # Initialize and train model
```

```

110     model = build_hybrid_model((X_train.shape[1], X_train.shape[2]))
111     history = model.fit(X_train, y_train,
112         batch_size=BATCH_SIZE,
113         epochs=EPOCHS,
114         validation_data=(X_val, y_val),
115         verbose=0,
116         callbacks=[EarlyStopping(patience=10,
117             restore_best_weights=True)])
118
119     # Evaluate performance
120     y_pred = (model.predict(X_val) > 0.5).astype(int)
121     accuracies.append(accuracy_score(y_val, y_pred))
122     precisions.append(precision_score(y_val, y_pred))
123     recalls.append(recall_score(y_val, y_pred))
124     f1_scores.append(f1_score(y_val, y_pred))
125     aucs.append(roc_auc_score(y_val, y_pred))
126
127     # Print cross-validation results
128     print("\nCross-validation Results:")
129     print(f"Average Accuracy: {np.mean(accuracies):.4f}±{np.std(accuracies):.4f}")
130     print(f"Average Precision: {np.mean(precisions):.4f}±{np.std(precisions):.4f}")
131     print(f"Average Recall: {np.mean(recalls):.4f} (±{np.std(recalls):.4f})")
132     print(f"Average F1-Score: {np.mean(f1_scores):.4f}±{np.std(f1_scores):.4f}")
133     print(f"Average AUC-ROC: {np.mean(aucs):.4f}±{np.std(aucs):.4f}")

```

A. 4. 8 AMCDD

```

1 import random
2 import pandas as pd
3 import numpy as np
4 import tensorflow as tf
5 tf.config.set_visible_devices([], 'GPU') # Disable GPU usage
6 import os
7 os.environ['HF_ENDPOINT'] = 'https://hf-mirror.com/' # Set HuggingFace mirror
8 from keras.models import Model
9 from keras.layers import Input, Dense, Bidirectional, LSTM, GRU, Dropout, GlobalAveragePooling1D
10 from keras.callbacks import EarlyStopping
11 from sklearn.model_selection import StratifiedKFold
12 from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_auc_score
13 import torch
14 from transformers import AutoTokenizer, AutoModel
15
16 # Set random seeds for reproducibility
17 tf.random.set_seed(42)
18 random.seed(42)
19 np.random.seed(42)
20
21 # Load dataset
22 data = pd.read_excel("./Dataset.xlsx")
23
24 # Initialize BERT model and tokenizer
25 tokenizer = AutoTokenizer.from_pretrained('nghuyong/ernie-3.0-base-zh')
26 model = AutoModel.from_pretrained('nghuyong/ernie-3.0-base-zh')

```

```
27
28 def extract_bert_embeddings(text, tokenizer, model, max_length=100):
29     """
30     Extract BERT embeddings from input text.
31
32     Args:
33         text (str): Input text
34         tokenizer: BERT tokenizer
35         model: BERT model
36         max_length (int): Maximum sequence length
37
38     Returns:
39         numpy.ndarray: Sentence embeddings in numpy format
40     """
41     inputs = tokenizer(text,
42                         return_tensors='pt',
43                         max_length=max_length,
44                         padding='max_length',
45                         truncation=True)
46
47     with torch.no_grad():
48         outputs = model(**inputs)
49
50     embeddings = outputs.last_hidden_state
51     return embeddings.squeeze().cpu().numpy()
52
53 # Generate BERT embeddings for all texts
54 X = np.array([extract_bert_embeddings(text, tokenizer, model, max_length=150) for text in data['text']])
55 y = data['class'].values # Target labels
56
57 def hybrid_model(input_shape):
58     """
59     Create a hybrid LSTM–GRU neural network model.
60
61     Args:
62         input_shape (tuple): Shape of input data
63
64     Returns:
65         keras.Model: Compiled Keras model
66     """
67     inputs = Input(shape=input_shape)
68     x = LSTM(256, return_sequences=True)(inputs) # LSTM layer with 256 units
69     x = Bidirectional(GRU(256, return_sequences=True))(x) # Bidirectional GRU layer
70     x = GlobalAveragePooling1D()(x) # Global average pooling
71     x = Dropout(0.3)(x) # Regularization
72     outputs = Dense(1, activation='sigmoid')(x) # Output layer
73
74     model = Model(inputs=inputs, outputs=outputs)
75     model.compile(
76         loss='binary_crossentropy',
77         optimizer='adam',
78         metrics=['accuracy']
79     )
```

```

80     return model
81
82 # Training parameters
83 BATCH_SIZE = 32
84 EPOCHS = 100
85
86 # Initialize metrics storage
87 accuracies, precisions, recalls, f1_scores, aucs = [], [], [], [], []
88
89 # Set up stratified 10-fold cross-validation
90 skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
91
92 for train_idx, val_idx in skf.split(X, y):
93     # Split data
94     X_train, X_val = X[train_idx], X[val_idx]
95     y_train, y_val = y[train_idx], y[val_idx]
96
97     # Build and train model
98     model = hybrid_model((X_train.shape[1], X_train.shape[2]))
99     history = model.fit(
100         X_train, y_train,
101         batch_size=BATCH_SIZE,
102         epochs=EPOCHS,
103         validation_data=(X_val, y_val),
104         verbose=0,
105         callbacks=[EarlyStopping(patience=10, restore_best_weights=True)])
106     )
107
108     # Evaluate model
109     y_pred = (model.predict(X_val) > 0.5).astype(int)
110     accuracies.append(accuracy_score(y_val, y_pred))
111     precisions.append(precision_score(y_val, y_pred))
112     recalls.append(recall_score(y_val, y_pred))
113     f1_scores.append(f1_score(y_val, y_pred))
114     aucs.append(roc_auc_score(y_val, y_pred))
115
116     # Print cross-validation results
117     print(f"Average Cross-Validation Accuracy: {np.mean(accuracies):.4f}±{np.std(accuracies):.4f}")
118     print(f"Average Cross-Validation Precision: {np.mean(precisions):.4f}±{np.std(precisions):.4f}")
119     print(f"Average Cross-Validation Recall: {np.mean(recalls):.4f}±{np.std(recalls):.4f}")
120     print(f"Average Cross-Validation F1 Score: {np.mean(f1_scores):.4f}±{np.std(f1_scores):.4f}")
121     print(f"Average Cross-Validation AUC-ROC: {np.mean(aucs):.4f}±{np.std(aucs):.4f}")

```

A. 5 参考书目

- [1] 李航. 机器学习方法 [M]. 北京: 清华大学出版社, 2022.
- [2] Snowflake Inc. *Streamlit: A faster way to build and share data apps*[EB/OL]. <https://streamlit.io/>.

致 谢

陇右故郡，兰州新府。大河明珠，丝路要枢。襟三秦而带五津，通西域以连卫藏。夷夏之交冲，兵家必争之地；山川之锁钥，商旅常往之途。景桓扬鞭，筑金城之雄塞；博望持节，启互市之鸿图。

余自皖入陇，溯淮水以渡黄河，经中原而穿秦岭。忆故园风物：庐州晓月映淮浦，广陵烟柳拂邗渠；凭栏回首处，江南雨细闻莺语，塞北风高听雁呼。

萃英山麓，修业四时。朝习六艺经传，暮览四序荣枯。皋兰耸峙，瞰百里华灯若星河倒泻；兴隆逶迤，踏千阶云径如阆苑清虚。尝出汉塞，越天山，观大漠孤烟之壮；亦临蜀地，游锦里，品市井烟火之殊。华夏泱泱，民风各异，博采众长而增识；俊采星驰，盛会相逢，切磋琢磨以相濡。建模析理，格物致知小试；穷经问道，学术初窥门间。实习躬行，察世间百态；静坐覃思，明心志所趋。至若夜半误车，徒步观星斗；更深码字，敲键伴蟾蜍。曾驱单车凌绝顶，亦宿旷野仰天衢。闲来吟诗作赋，兴至摄景描图。

弘毅之士，任重而致远；砥砺之行，功成在斯须。文理兼修夙愿酬，术业专攻志趣投。所遇皆善，幸甚至哉。椿萱恩深，养浩然之气以润德；冰玉质洁，继清白之风而慎修。程门立雪，承恩师刘公殷殷提携；杏坛传薪，得陈君师兄谆谆指引。定乾坤于方寸，启蒙科学门庭；探奥义于毫厘，指点迷津渡口。鹏飞子宸，砥柱中流，扶危济困，情深义重。更有习东轩懿诸君，谈经论道，潘江陆海各展才思；揽辔登高，霁月光风共谋新猷。后生可畏，颖脱囊锥，辩通今古，气贯虹霓。故交星散，德馨恒留，胸藏锦绣，志在鸿伟。或居蓟北，或客三吴，星分翼轸，地接荆湘。兰亭修禊，竹林纵酒，忝列群贤，如沐春柔。其余高情厚谊，虽竹帛难周，纸短情长，伏惟顿首。

外祖母氏，耄耋操劳，躬耕陇亩，笙磬谐鸣。断杼择邻之诚，虽未通经史；牵裾问字之切，竟如沐春风。青灯黄卷，常嘱人杰当为；粗茶淡饭，总期诗礼传家。恨彼苍穹不仁，摧萱堂于秋夜；痛游子兮未归，隔蓬山于重溟。忆当时明月，照空庭而悬孤影；念彼处清波，摇寒芦以诉哀声。水云乡里，犹闻机杼札札；松楸垄上，永驻慈颜茕茕。

时维槐月，序属暮春。细柳垂金缕，朱桥转画桡。白塔浮光，漫天花作雪；黄河涌浪，匝地气如潮。日衔西崦，霞染丹青成锦绣；月出东皋，辉凝碧落转琼瑶。游人载酒，踏歌恍入武陵渡；倦鸟投林，振翅犹追赤城标。俯仰百年，风云激荡。忆昔左衽遗忧，陆沉曾洒新亭泪；观今红旗漫卷，复兴正扬沧海桡。吾侪何幸，值此龙骧虎变之际；此身虽微，敢忘凤引九雏之昭？当效鲲鹏，绝云气以负青冥；更慕骅骝，踏霜蹄而驰紫霄。他日天涯重聚，当携兰芷，浮大白，倾肺腑于流觞曲水；今朝歧路将行，愿共肝胆，掣长鲸，照丹心于玉宇澄霄。

熏风拂槛，细雨跳珠。千嶂外，衡阳雁书。黄河九曲，临流觞咏。倚栏长啸处，孤城暮，天地庐。万里龙沙，星霜客途。忆往昔，少年意气：江南梦远，青衫红药；蕉窗夜雨，

驹影逝，墨云舒。

萃英高阁俯长川，玉笛吹残陇右烟。
画栋朝萦西域月，珠帘暮卷北庭天。
云移太华三秋树，浪拍昆仑九曲弦。
阁中墨卷今犹在，槛外河声送客船。

Our whole universe was in a hot dense state,

Then nearly fourteen billion years ago expansion started,
Wait!

The Earth began to cool;
The autotrophs began to drool;
Neanderthals developed tools;
We built a Wall;
We built the Pyramids;
Math science history unraveling the mystery,
That all started with the Big Bang.

Bang!

太常引·戍楼望月寄怀

冰轮碾破玉霄宫。银汉泻千重。羌笛咽清商，怅望处、云沙故封。
挑灯看剑，星槎横斗，铁甲带霜看。百战勒燕然，指顾间、狼烟尽删。

立夏时令，乙巳年四月初八，公历二零二五年五月五日，于积石堂