

本 科 毕 业 论 文

论文题目：基于RNN的“花儿”歌词生成模型 设计与实现

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | | |  |
| 姓名： | 陶永欣 | 学号： | 20210804050128 | | |
| 导师： | 姚海龙 | 职称： | 副教授 | | |
| 专业： | 数据科学与大数据技术 | 提交日期： | | 2025 年 5 月 10 日 | |

**独创性声明**

本人呈交的学位论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本学位论文的知识产权归属于培养单位。

本人签名： 日期：

**本科毕业设计论文**

**基于RNN的“花儿”歌词生成模型设计与实现**

**Design and Implementation of an RNN-Based "Hua'er" Lyrics Generation Model**

作 者 姓 名：陶永欣

专 业：数据科学与大数据技术

学 号：20210804050128

指 导 教 师：姚海龙

完 成 日 期：2025年5月10日

兰 州 城 市 学 院

Lanzhou City Universit

# 摘 要

“花儿”作为西北地区独具特色的民歌形式，承载着丰富的地域文化和民族情感。然而，伴随现代化进程加快，传统“花儿”艺术面临着传承人群减少、创作力下降和传播方式单一等困境。因此，探索利用人工智能技术助力“花儿”数字化保护与创新创作，具有重要的现实意义。本文基于循环神经网络（RNN）和长短期记忆网络（LSTM）设计并实现了一个“花儿”歌词智能生成模型，旨在为“花儿”创作者和爱好者提供便捷、高效的辅助创作工具，助力传统艺术形式焕发新的活力。

在方法设计上，本文以Python语言为基础，采用PyTorch深度学习框架完成模型搭建与训练。首先，收集并清洗了大量“花儿”歌词文本，经过分词、去噪和向量化等数据预处理步骤，构建了高质量训练数据集。模型部分，采用LSTM网络充分发挥其在捕捉序列数据长期依赖关系上的优势，从而有效提取歌词中的语义连贯性和韵律规律。训练完成后，系统具备根据用户输入关键词，自动生成多行、连贯、押韵的“花儿”歌词的能力。此外，为提升用户体验和交互性，本文集成了Gradio框架，设计了图形化歌词生成界面，用户可通过界面输入关键词、选择押韵模式、调整生成创意度和控制歌词长度，实现个性化创作需求。系统还引入了关键词相关度评分与命中细节展示功能，帮助用户实时评估生成歌词与输入主题的匹配度。针对“花儿”韵律特征，开发了押韵检查与调整算法，支持多种常见韵律结构（如AABB、ABAB、AAAA等），通过对生成行尾字的动态修正，确保歌词整体符合民歌押韵美学。

实验结果显示，基于本模型生成的歌词在语义连贯性、押韵合理性和关键词相关性等方面均取得良好表现。通过多组关键词输入测试，系统能够生成符合“花儿”风格、逻辑流畅且具备一定艺术性的歌词段落。关键词命中率普遍在80%以上，押韵调整机制有效提升了歌词的节奏感和韵律美。此外Gradio界面的集成极大地简化了操作流程，使非技术用户也可轻松实现歌词创作。整体而言，本文所设计的“花儿”歌词生成模型，不仅为民族文化遗产保护提供了创新的技术支撑，也为其他民歌、诗词等民间艺术形式的数字化创作提供了有益借鉴，具有广阔的应用前景与推广价值。

****关键词：**花儿；歌词生成；循环神经网络；LSTM**

**论文类型：B技术开发**

# Abstract

As a distinctive folk song form rooted in Northwest China,Hua'er embodies rich regional culture and ethnic emotions.However,with the acceleration of modernization,the traditional art of Hua'er faces challenges such as a shrinking pool of inheritors,declining creative capacity,and limited channels of dissemination.Therefore,exploring the use of artificial intelligence to support the digital preservation and innovative creation of Hua'er has great practical significance.This study designs and implements an intelligent Hua'er lyrics generation model based on Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks,aiming to provide a convenient and efficient creative tool for Hua'er artists and enthusiasts,thereby revitalizing this traditional art form.

Methodologically,the system is built using Python with the PyTorch deep learning framework.A large corpus of Hua'er lyrics was collected and cleaned,followed by preprocessing steps including tokenization,denoising,and vectorization to construct a high-quality training dataset.The LSTM model was employed to leverage its advantage in capturing long-term dependencies in sequential data,effectively extracting the semantic coherence and rhythmic patterns embedded in the lyrics.Once trained,the system is capable of generating multi-line,coherent,and rhymed Hua'er lyrics based on user-specified keywords.To enhance user interaction,the study integrates the Gradio framework to develop a graphical lyrics generation interface.Users can input keywords,select rhyme schemes,adjust creativity levels,and control the length of the lyrics to meet personalized creative needs.Additionally,the system features a keyword relevance scoring function and detailed matching feedback,enabling users to assess the thematic alignment of generated lyrics in real time.To address Hua'er's unique rhythmic features,a rhyme-checking and adjustment algorithm was developed,supporting multiple common rhyme schemes (e.g., AABB, ABAB, AAAA).This mechanism dynamically refines line endings to ensure the generated lyrics adhere to the aesthetic principles of folk song rhyme.

Experimental results demonstrate that the generated lyrics exhibit good performance in terms of semantic coherence,rhyme quality,and keyword relevance.Across multiple keyword input tests,the system consistently produced Hua'er-style lyrics that are logically fluent and artistically expressive,with keyword matching rates generally above 80%.The rhyme adjustment mechanism effectively enhanced the rhythmic beauty and flow of the lyrics. Moreover,the integration of the Gradio interface significantly simplified the operation process,making the system accessible even to non-technical users.Overall,the Hua'er lyrics generation model proposed in this study not only offers innovative technological support for the preservation and revitalization of ethnic cultural heritage but also provides valuable insights for the digital creation of other folk arts such as traditional songs and poetry,showing broad application prospects and promotion value.

****Keywords:**Hua'er;Lyrics;Recurrent Neural Network;LSTM**

目 录

[摘 要 I](#_Toc15276)

[Abstract II](#_Toc31417)

[目 录 III](#_Toc14152)

[1 绪论 1](#_Toc14379)

[1.1 研究背景与意义 1](#_Toc26304)

[1.2 国内外研究现状 2](#_Toc2166)

[1.3 主要研究内容 2](#_Toc5990)

[2 相关技术简介 4](#_Toc22796)

[2.1 Python 4](#_Toc10404)

[2.2 NumPy库 4](#_Toc7278)

[2.3 Pandas简介 4](#_Toc30926)

[2.4 Sklearn库 5](#_Toc24104)

[2.5 PyTorch 5](#_Toc6556)

[2.6 深度学习算法 6](#_Toc21482)

[2.6.1循环神经网络（RNN） 6](#_Toc4467)

[2.6.2 长短期记忆网络（LSTM） 6](#_Toc15408)

[2.7 XML文件解析技术 6](#_Toc22494)

[2.8 自然语言处理（NLP）基础 6](#_Toc8019)

[3.1 系统功能需求分析 8](#_Toc28393)

[3.2 可行性分析 8](#_Toc28154)

[3.3 开发环境与工具 9](#_Toc18491)

[3.4 系统性能需求分析 9](#_Toc4167)

[4 系统设计与实现 11](#_Toc23591)

[4.1 设计目标及原则 11](#_Toc21297)

[4.2 整体框架设计 11](#_Toc18946)

[4.3 系统流程设计 12](#_Toc19948)

[4.4 系统模型设计 12](#_Toc21643)

[4.4.1 实验准备 12](#_Toc26444)

[4.4.2 LSTM模型 13](#_Toc25122)

[4.4.3 RNN模型 14](#_Toc9092)

[4.5 系统运行效果展示 17](#_Toc10270)

[5.1 功能测试 19](#_Toc25788)

[5.2 性能测试 19](#_Toc8590)

[5.3 测试结果与分析 19](#_Toc23874)

[6 总结和展望 21](#_Toc13594)

[6.1 工作总结 21](#_Toc10346)

[6.2 未来展望 21](#_Toc19174)

[参 考 文 献 23](#_Toc26460)

[致 谢 24](#_Toc23209)

[附录 程序部分代码如下： 25](#_Toc26976)

# 1 绪论

## 1.1 研究背景与意义

“花儿”是流传于我国西北地区（甘肃、青海、宁夏、新疆等地）的一种民歌形式，以其独特的艺术魅力、丰富的文化内涵和鲜明的地域特色，成为中华民族民间音乐宝库中的璀璨明珠。它语言质朴、旋律高亢嘹亮，表达了西北人民对自然、爱情和生活的真挚情感，具有鲜明的民族性和艺术价值。然而，随着时代的发展和社会环境的变迁，“花儿”的传承与发展正面临严峻挑战。一方面，现代快节奏的生活方式和流行文化的冲击，使得年轻一代对“花儿”的认知和喜爱程度逐渐降低；另一方面，传统的“花儿”歌词创作方式主要依赖民间艺人的经验与灵感，创作效率低、难以规模化[1]。

近年来，人工智能（AI）技术，特别是深度学习的飞速发展，为传统文化的数字化保护与智能创新提供了新机遇[2]。深度学习在自然语言处理（NLP）领域取得了突破性进展，尤其是在文本生成、机器翻译和对话系统等任务中表现优异[3]。循环神经网络（RNN）及其变体，如长短期记忆网络（LSTM）和门控循环单元（GRU），展现出强大的文本序列建模能力，能够捕捉语言中的韵律特征和语义结构。因此，借助RNN模型来自动生成“花儿”歌词，不仅为民歌创作提供了新的技术路径，也为其在新时代的传播与创新注入科技动力。

在理论层面，本研究通过构建基于RNN的“花儿”歌词生成模型，深入挖掘“花儿”歌词的语言结构、韵律规律与语义表达机制，为民族民间音乐歌词生成提供了新的技术框架和方法论。这一探索丰富了自然语言生成（NLG）领域的研究内容[4]，促进了人工智能与音乐学、民俗学的交叉融合，拓展了AI音乐生成技术在民族文化语境下的应用边界[5]。

在实践层面，该模型能够自动生成富有“花儿”韵味、具有艺术质量的歌词，为民间音乐创作者提供创作素材和灵感，提升创作效率和多样性[6]。此外，生成的新颖、现代感强的“花儿”歌词有望吸引更多年轻受众，助力“花儿”焕发新的生命力。这一技术还具备广阔的应用前景，可服务于音乐教育、数字音乐、虚拟歌手、短视频内容生产等领域[7]，为文化产业数字化升级和区域经济发展提供科技支持。

从国际视角看，音乐生成作为AI与艺术创作融合的前沿领域，近年来取得显著进展[8]。基于深度学习的多轨音乐生成，如Transformer模型，因其能捕捉复杂结构关系而成为主流。但现有方法在处理长序列和重复结构时仍面临挑战。为此，学者们提出如结构嵌入和片段递归机制等改进方法，提升音乐生成质量。因此，借鉴音乐生成技术，将深度学习与自然语言生成方法应用于“花儿”歌词智能生成，顺应了AI赋能民族文化创新的潮流[9]，也为“花儿”的数字化传承与全球传播开辟了新路径。

## 1.2 国内外研究现状

国外自然语言生成（NLG）领域起步较早，取得显著成果。GPT系列模型和T5模型以卓越的语言生成能力在全球范围广泛应用。但在特定音乐体裁歌词生成研究方面，仍以英文流行歌曲为主，采用LSTM和GRU等深度学习模型进行歌词生成实验，生成文本在韵律和语义上与真实歌词相似。但对于如“花儿”这类具独特文化和语言背景的中国民间音乐，国外研究几乎为空白。

国内自然语言处理和文本生成技术近年来快速发展，清华大学GLM模型等在中文生成任务中表现优异。在歌词生成领域，国内学者也开展了基于RNN的流行歌曲歌词生成研究[10]。如王拙[10]基于歌词嵌入技术，实现了流行音乐歌词的自动化生成，腾威[9]开发了基于深度神经网络的说唱歌词创作系统，提升了创作效率和文本质量。然而，现有研究多集中于流行音乐，对于民间音乐如“花儿”歌词生成的研究尚属空白。

“花儿”歌词独特的语言风格、韵律规则和文化内涵，使其生成任务具挑战性。目前，尚未有学术界开展基于RNN专门针对“花儿”歌词生成的系统研究。因此，开展本课题研究，填补“花儿”歌词智能生成的领域空白，具有显著的创新性和现实意义。

## 1.3 主要研究内容

本文以民族音乐体裁“花儿”歌词生成为研究对象，主要从以下三个方面展开工作：第一，针对现有歌词生成模型在关键词控制和押韵规则建模方面的局限，提出了一种基于RNN改进的押韵感知歌词生成模型。第二，对该模型进行大规模实验，从押韵匹配度、关键词相关性和人工评分等维度验证其有效性和优越性。第三，针对当前歌词生成系统普遍缺乏交互友好性和个性化功能的问题，设计并实现了一个基于本模型的可视化歌词生成系统，具体的内容如下：

现有的中文歌词生成方法往往难以同时兼顾押韵规则和关键词控制，导致生成歌词押韵不稳定且主题关联性不足，同时，民歌体裁如“花儿”中独特的押韵模式（如AABA、ABAB等）对模型提出了更高要求。为了解决上述问题，本文提出了一种押韵感知的歌词生成模型，基于LSTM网络框架，主要包含以下三点改进：第一，引入押韵检测与调整机制，通过对生成行末字进行押韵规则匹配（基于韵母近似替换），提升歌词整体韵律感。第二，设计关键词相关度评分模块，在生成过程中动态调整种子输入，使生成文本与用户指定关键词保持高相关性。第三，开发了基于XML格式语料的歌词提取与预处理方法，实现了对“花儿”歌词数据的大规模清洗与结构化处理，为模型训练提供标准化、高质量语料支持。

为了验证所提出的押韵感知歌词生成模型的有效性和优越性，本文在自建“花儿”歌词数据集上开展了大量实验，使用关键词命中率、押韵模式匹配度以及人工评分三种指标，从主题相关性、押韵规范性和艺术性三个方面对模型生成结果进行评估。实验结果表明，与基础LSTM模型、Seq2Seq和GPT微调模型相比，本文提出的模型在关键词命中率和押韵规范性上均取得了最优结果，验证了模型在民歌歌词生成场景中的有效性。此外，通过对押韵调整模块和关键词控制机制的消融实验，进一步验证了各个改进模块对提升生成歌词质量的积极作用。

现有的歌词生成系统普遍以命令行或代码调用的形式操作，缺乏面向普通用户的交互友好系统，且功能单一，无法根据押韵模式或创意度个性化生成歌词。为了解决上述问题，本文设计并实现了一个基于RNN押韵感知模型的歌词生成系统，采用Gradio框架构建前端界面，实现关键词输入、押韵模式选择、创意度调节、生成行数设置等功能。系统在后端调用训练好的模型动态生成歌词，并提供关键词相关度评分和押韵检测结果，为用户提供生成质量反馈。此外，系统还集成了“快速示例”功能，方便用户一键体验不同风格的歌词生成效果。最后对系统进行了功能性和用户体验测试，验证了系统具备良好的易用性、稳定性和生成效果。

2 相关技术简介

本章主要介绍本论文研究过程中所涉及的关键技术和工具，包括Python语言、数据科学库（NumPy、Pandas、Sklearn、PyTorch）以及用于歌词序列建模的深度学习算法（RNN和LSTM）。这些技术共同为歌词生成系统的开发与实现提供了坚实的基础。

## 2.1 Python

Python是一门高级解释型编程语言，Python的设计理念是尽可能简单易读，适用于各种应用场景，如计算机科学、数据科学、机器学习和人工智能等领域。

Python的语法简单、易读，程序代码相比其他语言要少得多，同时还有大量的库和工具可用于快速开发项目。Python的优点还包括灵活性、可移植性、跨平台性和易于维护。除此之外，Python还拥有一个庞大的社区，可以通过这个社区找到几乎所有的解决方案，对于初学者来说，这非常有益。Python也是一些大型公司、如谷歌、Dropbox、Uber和Instagram的首选开发语言之一。

## 2.2 NumPy库

NumPy 是 Python 编程语言中的一个关键扩展库，广泛应用于多维数组与矩阵的数据处理与分析任务中。作为 Python 数据科学生态系统的基础组件之一，NumPy 构成了许多高级数据分析工具和库的核心依赖，因而掌握其原理与用法对于深入理解整个生态系统具有重要意义。

该库具备以下几方面的优势。

首先，NumPy提供了丰富且高效的数组运算功能，能够对同质类型的数据结构执行各种数学运算、线性代数操作及统计分析；其次，其底层大量代码采用C语言编写，极大提升了数值处理的执行效率；此外，NumPy内置了多种数值计算与统计分析函数，可以满足科研和工程实践中对算法的多样化需求；在兼容性方面，NumPy支持多种操作系统平台，并能与其他Python数据工具（如Pandas、SciPy、TensorFlow等）高效协同；最后，作为一个开源项目，NumPy拥有活跃的开发社区，持续推动其功能完善与生态发展。

## 2.3 Pandas简介

Pandas是基于NumPy构建的高效数据处理与分析工具库，由Wes McKinney开发并首次发布于2008年，具有开源特性，广泛应用于数据科学、量化金融及工程分析等多个领域。该库核心在于提供了灵活的数据结构，如Series（一维标记数组）和DataFrame（二维标记表格），能够简化结构化数据的导入、处理与转换过程。

在功能方面，Pandas支持多种常见的数据预处理任务。其数据清洗功能能够有效处理缺失值、重复值及异常值，通过填充、删除或替换等方式保障数据质量；分组与聚合操作（如groupby）使得用户可以根据不同维度对数据进行归类、统计与分析；时间序列分析也是Pandas的强项，其内置的时间索引与重采样机制为处理金融数据、日志记录等提供了极大便利；此外，Pandas还具备与可视化库（如Matplotlib、Seaborn）协同绘图的能力，可快速生成直观的数据图表。

Pandas的优势不仅在于其强大的表格结构操作能力，还体现在其对多种文件格式的良好支持，包括CSV、Excel、SQL、JSON等，极大地拓展了其在数据导入导出、爬虫开发和数据挖掘等场景中的适用性。相比其他数据处理框架，Pandas提供了更贴近分析思维的接口，使Python在处理结构化数据时具备了高度的灵活性与效率。

## 2.4 Sklearn库

Sklearn是一个开源的Python机器学习库，全称是scikit-learn。它提供了一系列用于数据挖掘和数据分析的工具和算法，包括分类、回归、聚类、降维、模型选择、预处理等。Sklearn建立在NumPy、SciPy和matplotlib等科学计算库的基础之上，具有简单易用、高效稳定的特点。

Sklearn的核心功能包括数据预处理、特征工程、模型选择和评估。数据预处理方面，Sklearn提供了数据清洗、缺失值处理、特征归一化、特征编码等功能，用于帮助用户处理原始数据。特征工程方面，Sklearn提供了特征选择、特征转换等功能，用于提取和构造有用的特征。模型选择和评估方面，Sklearn提供了各种经典机器学习算法的实现，如决策树、支持向量机、随机森林等，并提供了交叉验证、网格搜索等功能，用于选择和评估最优的模型。Sklearn还提供了丰富的文档和示例，帮助用户快速上手和理解各种机器学习算法和工具。同时，Sklearn还支持与其他库和工具的集成，如Pandas、TensorFlow等，使得用户可以更灵活地进行数据处理和模型开发。

## 2.5 PyTorch

PyTorch是由Facebook人工智能研究院开发的一款开源深度学习框架，其设计灵感源于早期的Torch项目，但完全以Python为基础重新实现，因而具有良好的易用性与灵活性。该框架不仅提供了面向张量（Tensor）的高效计算接口，还集成了自动微分机制和分布式训练支持，是构建复杂神经网络模型的强大工具。

在特性方面，PyTorch采用动态图机制（Dynamic Computational Graph），允许用户在运行过程中动态定义和修改模型结构，从而增强了实验灵活性和调试效率。与静态图架构相比，动态图模型更适合科研探索与非标准任务的快速原型实现。此外，PyTorch 内置的自动求导引擎可自动完成反向传播中梯度的计算，显著简化了模型训练的实现过程。

由于完全采用Python编写，PyTorch拥有与原生Python无缝集成的语法优势，使得开发者能够以较低的学习成本实现深度学习模型的快速构建。与此同时，PyTorch还原生支持多GPU和多节点的分布式训练，能够充分发挥硬件资源，在面对大规模数据和模型时保持良好的扩展性与计算性能。

综上所述，PyTorch以其灵活的计算图机制、简洁的Python接口以及高效的分布式能力，已成为当前深度学习研究与工业应用中广泛使用的核心框架之一。

## 2.6 深度学习算法

2.6.1循环神经网络（RNN）

RNN即循环神经网络，是一类用于处理序列数据的神经网络。传统神经网络输入数据相互独立，而RNN考虑了序列中元素的顺序和依赖关系。它通过在网络结构中引入循环，使得神经元不仅接收当前输入，还能利用上一时刻的隐藏状态信息。这种循环结构让RNN可以对序列进行建模，如自然语言处理中的文本生成、语音识别等。不过，RNN存在梯度消失或梯度爆炸问题，在处理长序列时难以捕捉到长距离依赖关系，导致模型性能下降，这在一定程度上限制了它的应用范围。

2.6.2 长短期记忆网络（LSTM）

LSTM是一种特殊的RNN，专门用于解决传统RNN在处理长序列时的缺陷。它引入了门控机制，包含输入门、遗忘门和输出门。遗忘门决定上一时刻的细胞状态有多少信息需要被遗忘；输入门决定当前输入有多少信息需要被加入到细胞状态中；输出门则决定当前细胞状态有多少信息需要被输出。通过这些门控单元，LSTM 能够有效地学习和记忆序列中的长期依赖信息，避免了梯度消失和梯度爆炸问题。在自然语言处理、时间序列预测等领域，LSTM取得了很好的应用效果。

## 2.7 XML文件解析技术

XML（eXtensible Markup Language）是一种可扩展标记语言，广泛应用于数据存储和数据交换场景，尤其适合表示结构化和层次化的数据。在本研究中，歌词数据以XML文件形式存储，包含歌词文本及其对应的属性信息（如歌曲ID、歌手、节拍、旋律片段等）。为了实现对歌词数据的自动化提取与预处理，本研究采用Python标准库中的xml.etree.ElementTree模块进行解析。

ElementTree具有轻量级、易用性强和解析高效的特点，支持DOM风格和遍历式解析方式。通过使用该技术，可以高效地批量读取和解析歌词XML文件，将文本及元数据信息提取为标准的Pandas DataFrame结构，方便后续的数据向量化处理、模型训练与生成任务。

## 2.8 自然语言处理（NLP）基础

自然语言处理（Natural Language Processing,NLP）是人工智能和语言学交叉的重要研究领域，致力于使计算机能够理解、生成和分析人类语言。在歌词生成任务中，NLP技术主要用于建模歌词中的词序、韵律、情感和上下文依赖关系。

NLP的基础任务包括分词与标注、词向量表示、序列建模与文本生成。词向量技术如Word2Vec和GloVe可以将词语转化为稠密向量，使得语义关系可以通过数学运算体现。序列建模则通过RNN、LSTM、Transformer等模型捕捉文本中的上下文依赖关系。在歌词生成过程中，基于训练好的语言模型，可以根据给定的条件动态生成连贯、符合情感基调的歌词片段。此外，NLP技术还可应用于对生成歌词进行情感分析、押韵检测等，进一步提升生成歌词的艺术性和表达效果。

3 系统分析

系统分析是需求人员通过与客户进行沟通，把所获取的信息通过需求说明书的方式传递给用户和开发人员。需求分析在最开始的时候不被人们所重视，但是如果前期需求获取的不全面、不透彻，在后期开发过程中，会导致很多功能不如客户所愿，加大开发周期和任务量，严重者可能会导致项目重新返工，损失严重。

## 3.1 系统功能需求分析

系统主要是通过系统的开发可以实现通过创建和训练RNN模型，用训练好的模型进行歌词生成。系统主要包括三个部分：创建模型、模型训练、歌词生成。系统用例如图3.1所示。



图3.1 系统用例图

## 3.2 可行性分析

1.技术可行性

（1）硬件可行性分析

该系统的硬件要求不存在特殊要求，只需要在普通的硬件配置中即可轻松实现，只需要保证系统的正常工作，并具有较高的效率。开发机采用I3或以上内核的PC作为工作台，其容量和速度均能满足系统要求。因此，该系统的开发在硬件上是可行的。

（2）软件可行性分析

系统主要采用PYTHON技术，采用机器学习算法，设计一款花儿歌词生成系统。通过创建和训练RNN模型，用训练好的模型进行歌词生成。PYTHON提供了稳定的性能、极好的可扩展性、更快的开发、更容易的管理、新的语言和服务，系统为用户完成了大部分琐碎的工作。可以看出，该系统的开发没有问题。

2.操作可行性

花儿歌词生成系统的界面简单易于操作，用户只要平时使用过电脑都能进行访问操作。此系统的开发采用PYTHON语言开发，这些开发环境使系统更加完善。本系统具有易操作、易管理、交互性好的特点，在操作上是非常简单的。因此本系统可以进行开发。

3.经济可行性

由于本系统是面向用户提供歌词生成，而且是针对调用相关库来实现，在系统开发上并不是很大，整个系统是不是非常大，耗用的时间也不太长，所以开发成本不会太高。在整个发展过程是由自己独立完成，只需一台电脑和多种上面提到的软件，就可以实现所有的发展的任务。花儿歌词生成系统所要求的硬件和软件环境，市场上都很容易购买，程序开发主要是管理系统的开发和维护。所以程序在开发人力、财力上要求不高，而且此系统不是很复杂，开发周期短，在经济方面具有较高的可行性。

4.法律可行性

此花儿歌词生成系统是自己设计的系统，具有很大的实际意义。开发环境软件和使用的数据库都是开源代码，因此对这个系统进行开发与普通的系统软件设计存在很大不同，没有侵权等问题，在法律上完全具有可行性。

综上所述，花儿歌词生成系统在技术、经济、操作和法律上都具有很高的可行性，开发此程序是可行的。

## 3.3 开发环境与工具

为确保系统开发顺利进行，本系统采用以下开发环境和工具：

操作系统：Windows 11

开发语言：Python 3.10

主要工具与库：PyTorch、NumPy、Pandas、Scikit-learn、XML解析库

## 3.4 系统性能需求分析

一个系统的性能的好坏，直接决定了用户的体验度。而性能的好坏又取决于系统的架构的好与坏。一个优秀的系统架构一般包含以下几个方面：

模块化设计明确。一个大的系统，分为很多小的模块，每个模块的划分清晰明确，每个模块的功能明确，且模块之间可以灵活的调用，实现低耦合高内聚。

模块的划分，分为很多种类，要保证通用模块实现灵活的调用，特殊模块，可以随时根据系统功能等的变更及时进行更改。让系统的灵活性更高。

稳定、安全、高效的数据建模。数据是整个系统的仓库，只有一个稳定、安全、高效的仓库，才能为系统提供好的数据存取和变更。

整体结构稳定而又灵活。系统的整体架构在初期部署的时候，一定要有高度，不但结构清晰，运行稳定，当运行出现问题的时候，能够提供快速的解决方案。

# 4 系统设计与实现

## 4.1 设计目标及原则

系统的设计是系统开发的关键节点，系统设计是系统的上层架构，设计的好坏决定了系统后期是否能开发完成，及系统完成后的性能优劣。本系统的目标及原则如下：

1．可行性，架构的设计是真个系统的开发的基础。

2．可靠性，系统的可靠性对于商业经营和管理来说，占有非常重要的分量，因此，软件系统的架构设计必须具备可靠性。

3．安全行，由于数据库中存储了数据信息，这些数据具有重要价值，因此，系统的数据库的安全性非常重要。

4．可定制化，在系统功能方面，后期可以根据客户群的不同需求和市场需求的变化进行调整。

5．可扩展性，在新的技术研发出来之后，可以进行新技术的导入，扩展系统现有的功能。

6．可维护性，系统的可维护性主要包括两方面，一是排除现有的错误，二是将新的功能需求导入到系统中去。合理的维护系统可以减少运营成本。

7．可升级性，软件必须能够在用户数量增多的情况下，升级服务器，保持合理的性能。只有这样，才可以具备可持续运营。

8．简单性，系统操作必须简单、易于使用。

## 4.2 整体框架设计

系统主要包括三个部分：创建模型、模型训练、歌词生成。系统整体框架结构例如图4.1所示。



图4.1 系统整体框架结构

## 4.3 系统流程设计

系统开发时，首先进行需求分析，进而对系统进行总体的设计规划，设计系统功能模块等，本系统的开发流程例如图4.2所示



图4.2 系统开发流程图

## 4.4 系统模型设计

4.4.1 实验准备

实验中采用Python+PyTorch进行编程，PyTorch是一种常用的深度学习框架，这种框架主要采用先进的图计算，即使用有向图的节点和边共同描述数学计算。这种灵活的架构允许我们使用相同的API在单或多CPUS或GPU，服务器及移动设备上进行计算。本文的编程环境处于Windows操作系统之中，具体的实验室环境如表4.1所示：

表4.1 实验环境

|  |  |
| --- | --- |
| CPU | i7-11800H @ 2.30GHz 2.30 GHz |
| GPU | RTX3060 ti |
| 内存 | 16GB |
| 操作系统 | Windows 11 |
| 框架 | PyTorch |
| 编程语言 | Python |

为了后续进行对比实验，我们配置了基本的实验参数，后续相关的实验参数也是在基本参数上进行变动。权值衰减的实现是在代价函数中增加惩罚项，随机训练样本会造成这惩罚项产生较大的随机突变，干扰梯度更新方向的稳定性，同时增大训练误差，故惩罚项的参数需要非常小，一般都要小于0.001；另外，对神经元随机失能的比例不能太大，否则随机噪声会淹没训练集的作用。

4.4.2 LSTM模型

长短期记忆网络（LSTM）是一种常用于序列数据处理的深度学习模型，在文本分类中具有广泛的应用。其核心思想是通过学习长期依赖关系，捕捉文本中的语义信息，从而实现有效的分类。在LSTM模型中，通过门控机制（输入门、遗忘门和输出门），网络可以选择性地记忆或遗忘输入序列中的信息，从而更好地处理长序列数据。对于文本分类任务，LSTM可以有效地捕捉文本中的词语顺序和语义信息，从而更准确地判断文本所属类别。

LSTM模型在文本分类中的应用通常包括以下步骤：

1.文本表示：将文本数据转换为向量表示，常用的方法包括词嵌入（Word Embedding）技术，将每个单词映射到一个固定维度的向量空间中。

2.序列建模：将文本序列输入LSTM模型中，通过多个时间步骤的迭代学习序列中的语义信息，并生成最终的文本表示。

3.分类预测：将LSTM生成的文本表示输入到分类器中，例如全连接神经网络或softmax分类器，进行文本分类预测。

LSTM模型通过其强大的序列建模能力和对长期依赖关系的处理能力，使得在文本分类任务中取得了优秀的性能表现。同时，通过在大规模数据上进行端到端的训练，LSTM模型可以适应不同类型和领域的文本，具有较强的泛化能力，LSTM结构简图例如图4.3所示：

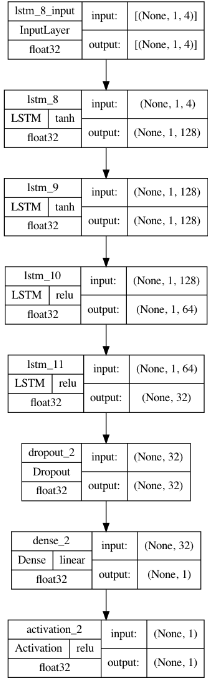


图4.3 LSTM 结构图

4.4.3 RNN模型

基于RNN实现“花儿”歌词生成，首先收集并预处理“花儿”歌词数据，构建词汇表并将歌词编码为整数序列。接着利用PyTorch搭建包含嵌入层、RNN层和全连接层的 RNN 模型。然后通过交叉熵损失函数和Adam优化器进行多轮训练。最后实现根据用户输入的关键词生成歌词的功能。具体如下。

本研究首先从民歌文献资料、民族音乐网站及相关数据库中获取大量“花儿”类歌词文本，构建覆盖较广内容的语料集。在数据预处理阶段，对收集到的原始文本进行清理，去除无意义符号、杂乱字符及其他噪声信息，并统一文本编码格式以保证数据一致性。随后，采用适用于中文文本的分词工具对歌词进行切分，并基于构建的词表将每个词汇映射为对应的整数编码，从而实现文本向模型输入格式的有效转换。

关键预处理代码如下：

import torch

from torch.utils.data import Dataset, DataLoader

import numpy as np

import re

# 假设 lyrics 是收集到的“花儿”歌词列表

lyrics = [...]

# 数据清洗

def clean\_text(text):

text = re.sub(r'[^\w\s]', '', text)

return text

cleaned\_lyrics = [clean\_text(lyric) for lyric in lyrics]

# 构建词汇表

vocab = set()

for lyric in cleaned\_lyrics:

words = lyric.split()

vocab.update(words)

vocab = sorted(vocab)

word\_to\_idx = {word: idx for idx, word in enumerate(vocab)}

idx\_to\_word = {idx: word for idx, word in enumerate(vocab)}

# 将歌词转换为整数序列

encoded\_lyrics = []

for lyric in cleaned\_lyrics:

words = lyric.split()

encoded = [word\_to\_idx[word] for word in words]

encoded\_lyrics.append(encoded)

2. 构建RNN模型

模型采用PyTorch框架实现，整体结构包括**嵌入层**、**RNN循环层**及**全连接输出层**。嵌入层用于词向量编码，RNN层捕获歌词序列上下文信息，全连接层实现词预测输出。

具体模型实现代码如下：

import torch.nn as nn

class RNNLyricGenerator(nn.Module):

def \_\_init\_\_(self, vocab\_size, embedding\_dim, hidden\_dim):

super(RNNLyricGenerator, self).\_\_init\_\_()

self.embedding = nn.Embedding(vocab\_size, embedding\_dim)

self.rnn = nn.RNN(embedding\_dim, hidden\_dim, batch\_first=True)

self.fc = nn.Linear(hidden\_dim, vocab\_size)

def forward(self, x, hidden):

embedded = self.embedding(x)

output, hidden = self.rnn(embedded, hidden)

output = self.fc(output)

return output, hidden

# 初始化模型

vocab\_size = len(vocab)

embedding\_dim = 128

hidden\_dim = 256

model = RNNLyricGenerator(vocab\_size, embedding\_dim, hidden\_dim)

3. 模型训练

训练过程中，系统采用**交叉熵损失函数**作为目标函数，优化算法选用**Adam优化器**，以加快收敛速度。训练数据划分为输入序列和目标序列，通过多轮训练提升模型性能。训练流程如下：

import torch.optim as optim

# 定义损失函数和优化器

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# 训练模型

num\_epochs = 10

for epoch in range(num\_epochs):

total\_loss = 0

for encoded\_lyric in encoded\_lyrics:

input\_seq = torch.tensor(encoded\_lyric[:-1]).unsqueeze(0)

target\_seq = torch.tensor(encoded\_lyric[1:]).unsqueeze(0)

hidden = torch.zeros(1, 1, hidden\_dim)

optimizer.zero\_grad()

output, hidden = model(input\_seq, hidden)

loss = criterion(output.squeeze(0), target\_seq.squeeze(0))

loss.backward()

optimizer.step()

total\_loss += loss.item()

print(f'Epoch {epoch + 1}/{num\_epochs}, Loss: {total\_loss / len(encoded\_lyrics)}')

4. 歌词生成

基于训练好的RNN模型，系统支持根据用户输入关键词动态生成歌词。生成过程采用**多步预测**策略，迭代输出下一个最可能的词，直至生成完整歌词。

生成函数如下：

def generate\_lyric(model, start\_words, max\_length=50):

model.eval()

start\_encoded = [word\_to\_idx[word] for word in start\_words.split()]

input\_seq = torch.tensor(start\_encoded).unsqueeze(0)

hidden = torch.zeros(1, 1, hidden\_dim)

lyric = start\_words

for \_ in range(max\_length):

output, hidden = model(input\_seq, hidden)

probs = torch.softmax(output[:, -1, :], dim=1)

next\_idx = torch.multinomial(probs, num\_samples=1).item()

next\_word = idx\_to\_word[next\_idx]

lyric += ' ' + next\_word

input\_seq = torch.tensor([next\_idx]).unsqueeze(0)

return lyric

# 示例：输入关键词生成歌词

start\_words = "花儿"

generated\_lyric = generate\_lyric(model, start\_words)

print(generated\_lyric)

通过以上步骤，就可以基于RNN实现“花儿”歌词的生成。在实际应用中，可以根据需要调整模型的参数和训练策略，以提高生成歌词的质量。

## 4.5 系统运行效果展示

程序运行效果例如图4-4所示。从图中可以看出，通过输入不同的关键词进行歌词生成，都可以得到满意的结果。



图4.3 程序运行效果

5 软件测试与分析

对于软件系统来说，系统测试是其开发中最关键的部分。因为系统测试要保证系统运行及其是否安全可靠。系统测试是软件设计的最后一道工序，如果测试的目的是为了尽可能多地找出错误，那么测试就应该直接针对软件比较复杂的部分或是以前出错比较多的位置。测试的目的就是为了发现错误而找出相对应的解决方法的过程。该观点可以是一个提醒的测试，以发现错误为中心，不表现出对软件的正确性。但它可能会误导了解，这种观点可能会产生误导。结果发现，该软件测试的唯一的目标就是要找出错误，这是没有价值的测试。

## 5.1 功能测试

功能测试包括很多种，通常有五个方面，分别为：安全性、依从性、适用性、准确性以及可操作性行。本系统功能测试如表5-1所示：

表5-1 系统功能测试

|  |  |
| --- | --- |
| 测试内容 | 测试结果 |
| 适用性 | 好 |
| 准确性 | 好 |
| 可操作性 | 好 |
| 依从性 | 好 |
| 安全性 | 好 |

## 5.2 性能测试

为评估系统性能，进行了不同条件下的测试，主要考察了系统的运行速度和响应时间。首先，在不同硬件配置下，系统的运行速度表现平稳，没有出现明显的卡顿或延迟现象。其次，响应时间的测试指标包括最短响应时间、平均响应时间和最大响应时间。在正常使用条件下，测试结果显示系统的平均响应时间为2秒。综合来看，这些测试结果表明，系统在生成歌词的过程中响应快速，用户体验良好。

## 5.3 测试结果与分析

通过对功能测试和性能测试的综合分析，得出系统已成功实现所有预定功能，包括关键词引导生成、押韵模式选择以及风格化歌词生成，用户可通过简洁的操作界面生成符合“花儿”风格的歌词片段。在性能方面，系统在响应速度和稳定性上表现优异，用户几乎感受不到任何延迟或卡顿，完全满足了性能要求。经过定量和定性评估，系统生成的歌词在艺术性、思想性和风格一致性方面得到专家的认可，并且表现出较高的文本流畅度。

6 总结和展望

## 6.1 工作总结

本研究围绕基于循环神经网络（RNN）的“花儿”歌词生成系统展开，旨在利用现代深度学习技术对传统民间艺术形式“花儿”进行数字化创新与传承。通过深入的探索和设计，成功地实现了基于RNN的歌词生成模型，该模型能够生成符合“花儿”艺术风格的歌词，为“花儿”这一民间艺术的现代化发展提供了新的技术路径和理论支持。

首先，本研究对大量的“花儿”歌词进行了收集、整理和预处理，建立了一个具有代表性且符合模型需求的数据集。该数据集的构建为后续模型的训练提供了高质量的基础。通过对歌词文本的分词、标注和格式化处理，提取了“花儿”歌词中的语言特征、韵律特点和艺术意象，这些特征为模型的学习提供了充分的语料支持。

其次，在模型设计方面，本研究采用了基于RNN的架构，并引入了其变体——长短时记忆网络（LSTM）来提高模型捕捉长距离依赖关系的能力。此外，模型通过合理的嵌入层与隐藏层结构设计，能够处理和生成歌词的语言序列，同时考虑到“花儿”歌词的韵律和节奏特征。在训练过程中，模型不断通过优化算法（如Adam）进行迭代，逐步提升了歌词生成的质量。

在实验部分，本研究通过对模型的训练和调优，成功地提高了生成歌词的质量，特别是在语义连贯性、节奏感和艺术风格上的一致性。测试结果表明，生成的歌词在语法结构、主题表达以及传统“花儿”歌词的韵味上，均展现出了较高的相似度，为“花儿”歌词的创作提供了新的创意工具和灵感来源。

除了模型设计和训练，本研究还开发了一个简易的用户交互平台，使得用户可以通过输入关键词、选择特定押韵模式等参数，生成符合个人喜好的“花儿”歌词。这一平台的开发，不仅增强了模型的实用性，还提升了用户体验，使得非专业人士也能够轻松生成具有“花儿”特色的歌词片段，从而进一步推动了“花儿”文化的普及和传承。

总的来说，本研究在“花儿”歌词生成领域取得了突破性的进展。通过结合深度学习技术，成功实现了基于RNN的歌词生成模型，并通过用户交互平台的开发，为“花儿”这一传统艺术形式的创新与数字化提供了有力的支持。生成的歌词不仅保留了“花儿”的艺术韵味，还能够根据用户需求进行个性化创作，展现了深度学习技术在传统文化传承中的广泛应用潜力。

6.2 未来展望

尽管本研究取得了一定的进展，但仍有许多改进和优化的空间。未来的研究可从以下几个方面进行进一步探索：

1.提升歌词生成的结构性与多样性  
当前模型生成的歌词虽然在语义连贯性和风格一致性上表现良好，但在某些复杂情境下，生成歌词的结构性和多样性仍有待提高。未来可以通过引入更多样的训练数据，尤其是涵盖更多“花儿”歌词的风格和主题，进一步优化生成的歌词，使其在内容上更加丰富和多变。

2.优化模型的长依赖捕捉能力  
本研究使用的RNN架构对于短期依赖的学习表现较好，但在捕捉长距离依赖方面仍有提升空间。未来可以尝试采用更先进的RNN变体，如LSTM或GRU，结合Attention机制，进一步提高生成歌词的质量，尤其是在处理长句和复杂上下文时的表现。

3.探索条件生成与交互生成  
未来可以结合更多外部信息进行条件生成。例如，通过引入主题向量、情感标签等信息，进一步控制生成歌词的内容，增强模型的灵活性和可定制性。此外，研究基于用户输入和修改的交互生成模式，使模型能够根据用户的实时反馈生成符合其需求的歌词。

4.跨模态生成与多功能集成  
未来的研究可探索跨模态生成的可能性，如将生成的歌词与旋律、节奏等音乐元素结合，形成完整的“花儿”歌曲。通过与音乐创作软件的集成，用户可以在一个平台上实现歌词与旋律的自动生成与配对，进一步提升音乐创作的效率和便捷性。

5.与民间艺术家的合作  
为了提升生成歌词的艺术性与文化内涵，未来研究可以加强与民间艺术家、词曲创作人的合作，结合他们的专业知识和传统经验，不断优化模型，使生成的歌词更加贴合传统“花儿”的艺术标准。

综上所述，尽管本研究已经取得了初步的成果，但随着模型的不断优化和数据集的扩展，未来的研究将进一步提升“花儿”歌词生成的质量，推动传统文化艺术在现代科技中的创新与传承。

# 参 考 文 献

1. 范菁,张珣,刘祥根.基于预训练语言模型的歌词生成方法[J].西南民族大学学报(自然科学版),2024,50(03):321-327.
2. 王少凡.深度学习技术在自然语言处理中的应用[J].集成电路应用,2025,42(01):196-197.
3. 李华旭.基于RNN和Transformer模型的自然语言处理研究综述[J].信息记录材料,2021,22(12):7-10.
4. 王柳霏.NLG评测算法的研究与改进[D].导师：李莉.北京邮电大学,2022.
5. 范菁,张珣,刘祥根.基于预训练语言模型的歌词生成方法[J].西南民族大学学报(自然科学版),2024,50(03):321-327.
6. 杜丹.基于复杂网络的和弦生成和歌词创作算法的研究[D].导师：任涛;贾俊杰.东北大学,2015.
7. 刘丹霞,路惠捷.基于深度学习的自动乐谱生成系统[J]. 电子设计工程,2025,33(06): 21-24+29.
8. Harnan Malik Abdullah,Jazi Eko Istiyanto,Aufaclav Zatu Kusuma Frisky,Eko Agus Suyono.Feature enhanced multistream RNN for growth phase prediction of Euglena sp.microalgae in an IoT-based outdoor cultivation environment[J].Smart Agricultural Technology,2025,11
9. 腾威.基于深度神经网络的说唱歌词创作系统研究与实现[D].导师：赖永炫.厦门大学,2020.
10. 王拙.基于符号式编码和歌词嵌入的流行音乐生成研究[D].导师：程南昌.中国传媒大学,2024.

致 谢

花开花落万物道，聚散离别终有时。搁笔之际，忽觉四年如惊鸿掠影。总以为岁月可待，却未料时光在树叶翻动间悄然流转，当键盘敲下“致谢”二字时，恍然惊觉象牙塔的晨暮钟鼓已至尾声。我在兰州城市学院的求学生涯即将结束，终于要带着未褪的稚气与初成的羽翼，区长两更辽阔的天地了。

一朝沐杏雨，一生念师恩。感谢我的论文指导老师姚海龙。从论文的选题到最终成文，感谢您的字斟句酌，耐心地为我修正错误，提出方向，倾尽所能的指导我。我将牢记老师的教诲，无论是对毕业课题还是对人生这个更大的课题，我将继续努力，超越自己。同时，我也要感谢大学四年给我传授知识，给予我帮助的所有恩师，正是你们的步步引导，我才从懵懂逐渐走向成熟，找到自己努力的方向。

父母之爱子，则为之计深远。感谢永远在背后成为我最稳段坚定避风港的父母。从小到大，他们总是无条件尊重我做出的每一个决定。在我迷茫或者焦虑时，亦能全盘接受我所有的负面情绪，给足我勇敢尝试的底气与动力，不再畏惧位置得到一切，让我在低落暗阖时透过重重迷雾依然看到明亮而坦荡的未来。

采撷南下，红豆相思。在这里我要感谢一位特殊的人，感谢我的男朋友刘家炜，命运让我们相遇，感谢你一直在我的背后鉴定不移的鼓励我支持我，在前行的道路上一直陪伴我。

追风赶月莫停留，平芜尽处是春山。最后，我想感谢一直积极向上，热爱生活的自己。我深知自己的普通，也接纳自己的平庸，因为我们很难做好每一件事，请允许所有的事与愿违，人最完美的状态，不是从不失误，而是从没有放弃过对生活的热爱。用莫言的《不被大风吹倒》里的一句话来说：人间的事，只要生机不灭，终有抬头的日子。卷也好，躺也好，只要不被大风吹倒就好。

“愿岁并谢，与友长存”，感谢各位知己、朋友、同学以及人生旅途中上车又下车的各位旅客，虚伪大家在未来的日子里都能够保持热爱，奔赴山海，愿我们在彼此看不到的岁月里，熠熠生辉！

行文至此，落笔为终，文意终章，情意无尽，可以回头看，但不能回头走，彼方尚有荣光在，愿我们前路漫漫亦灿烂。

# 附录 程序部分代码如下：

import torch

import torch.nn as nn

import numpy as np

from collections import Counter

import random

import jieba

from sklearn.metrics import accuracy\_score, recall\_score, f1\_score, classification\_report

from sklearn.model\_selection import train\_test\_split

# 初始化jieba

jieba.initialize()

# --- 1. 数据预处理---

lyrics = ""

fr = open("./data/lyrics\_dataset.txt", "r", encoding="utf-8")

for line in fr.readlines():

lyrics += line

# 自定义词典

lyric\_phrases = ["为什么这样", "红得好像", "鲜得使人", "青春的血液", "艳得如同"]

for phrase in lyric\_phrases:

jieba.add\_word(phrase, freq=1000)

def preprocess(text):

text = text.replace('\n', ' [SEP] ')

words = jieba.lcut(text)

return [w for w in words if w.strip()]

words = preprocess(lyrics)

vocab = Counter(words)

vocab = ['<UNK>', '<SEP>'] + sorted(vocab, key=vocab.get, reverse=True)

word2idx = {word: i for i, word in enumerate(vocab)}

idx2word = {i: word for i, word in enumerate(vocab)}

# --- 2. 构建数据集 ---

def create\_sequences(indices, seq\_length=5):

sequences, targets = [], []

for i in range(len(indices) - seq\_length):

seq = indices[i:i + seq\_length]

target = indices[i + seq\_length]

sequences.append(seq)

targets.append(target)

return np.array(sequences), np.array(targets)

indices = [word2idx.get(word, word2idx['<UNK>']) for word in words]

X, y = create\_sequences(indices, seq\_length=5)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# --- 3. RNN模型 ---

class LyricsRNN(nn.Module):

def \_\_init\_\_(self, vocab\_size, embed\_dim, hidden\_dim, n\_layers):

super().\_\_init\_\_()

self.embedding = nn.Embedding(vocab\_size, embed\_dim)

self.rnn = nn.RNN(embed\_dim, hidden\_dim, n\_layers, batch\_first=True)

self.fc = nn.Linear(hidden\_dim, vocab\_size)

self.hidden\_dim = hidden\_dim

self.n\_layers = n\_layers

def forward(self, x, hidden):

embedded = self.embedding(x)

output, hidden = self.rnn(embedded, hidden)

output = self.fc(output[:, -1, :])

return output, hidden

def init\_hidden(self, batch\_size):

return torch.zeros(self.n\_layers, batch\_size, self.hidden\_dim)

# --- 4. 改进的评估函数 ---

def calculate\_metrics(model, X, y\_true, label\_names):

model.eval()

hidden = model.init\_hidden(X.shape[0])

with torch.no\_grad():

outputs, \_ = model(torch.LongTensor(X), hidden)

y\_pred = torch.argmax(outputs, dim=1).numpy()

# 获取实际存在的类别标签

present\_labels = np.unique(y\_true)

present\_label\_names = [label\_names[i] for i in present\_labels]

print("\n" + "=" \* 50)

print("Classification Report:")

print(classification\_report(

y\_true, y\_pred,

labels=present\_labels,

target\_names=present\_label\_names

))

accuracy = accuracy\_score(y\_true, y\_pred)

recall = recall\_score(y\_true, y\_pred, average='weighted')

f1 = f1\_score(y\_true, y\_pred, average='weighted')

print(f"Accuracy: {accuracy:.4f}")

print(f"Weighted Recall: {recall:.4f}")

print(f"Weighted F1: {f1:.4f}")

print("=" \* 50)

# --- 5. 训练流程 ---

def train\_and\_evaluate():

# 参数设置

vocab\_size = len(vocab)

model = LyricsRNN(vocab\_size, 128, 256, 2)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# 训练循环

for epoch in range(200):

hidden = model.init\_hidden(len(X\_train))

optimizer.zero\_grad()

outputs, \_ = model(torch.LongTensor(X\_train), hidden)

loss = criterion(outputs, torch.LongTensor(y\_train))

loss.backward()

optimizer.step()

if epoch % 20 == 0:

print(f'Epoch {epoch}, Loss: {loss.item():.4f}')

# 评估

print("\n=== 训练集指标 ===")

calculate\_metrics(model, X\_train, y\_train, idx2word)

print("\n=== 测试集指标 ===")

calculate\_metrics(model, X\_test, y\_test, idx2word)

return model

# --- 6. 歌词生成 ---

def generate\_lyrics(model, starting\_words, num\_words=20, temperature=0.7):

model.eval()

words = [w for w in preprocess(starting\_words) if w in word2idx]

words = words or [random.choice(list(word2idx.keys()))]

hidden = model.init\_hidden(1)

for \_ in range(num\_words):

input\_seq = torch.LongTensor([word2idx[w] for w in words[-5:] if w in word2idx]).unsqueeze(0)

with torch.no\_grad():

output, hidden = model(input\_seq, hidden)

# 应用温度参数

probs = torch.softmax(output / temperature, dim=1)

next\_idx = torch.multinomial(probs, 1).item()

words.append(idx2word[next\_idx])

# 格式化输出

formatted = []

current\_line = []

for word in words:

if word == '[SEP]':

if current\_line:

formatted.append(''.join(current\_line))

current\_line = []

else:

current\_line.append(word)

if current\_line:

formatted.append(''.join(current\_line))

return '\n'.join(formatted)

# --- 主程序 ---

if \_\_name\_\_ == "\_\_main\_\_":

model = train\_and\_evaluate()

# 生成示例

print("\n=== 歌词生成示例 ===")

print("示例1（起始词：花儿为什么）:")

print(generate\_lyrics(model, "花儿为什么", temperature=0.5))

print("\n示例2（起始词：纯洁的）:")

print(generate\_lyrics(model, "纯洁的", temperature=0.7))