FNLP Assignment 4 Project 1 Report

Sub-task 1

在仅提供一本壮语语法书条件下，将指定的简单壮语句子准确地翻译成中文。

---

1. 实现方法

为应对这一挑战，本项目设计并实现了一套基于检索增强生成（Retrieval-Augmented Generation, RAG）的高度模块化翻译框架。该框架的核心思想是，在调用大语言模型（LLM）API（`qwen-max`）之前，先通过一系列精密的检索和组织策略，为模型动态构建一个信息极其丰富的提示（Prompt），从而最大限度地激发其在未知语言上的“在上下文学习”（In-Context Learning）能力。

1.1 代码结构

为了保证实验的灵活性和可复现性，我们的所有实现都整合在一个名为`final.py`的脚本中，并采用了高度模块化的设计。

- **配置管理:** 我们使用 `tap` 库定义了一个 `Args` 类，用于集中管理所有的超参数和配置开关，例如，是否启用检索（`use_retrieval`）、是否启用自我修正（`use_self_correction`）、检索模型的名称（`embedding_model_name`）以及混合检索的权重（`hybrid_alpha`）等。这使得我们可以通过命令行轻松地进行多种实验配置的组合与比较。
- **核心函数模块:** 系统的关键功能被拆分为一系列独立的函数，每个函数职责单一，便于维护和升级。主要包括：
    - `load_embedding_model`: 负责加载并缓存语义向量模型。
    - `find_best_rule` & `find_best_examples`: 负责从知识源中进行信息检索。
    - `build_prompt`: 负责将检索到的信息“装配”成最终的提示。
    - `get_qwen_max_translation`: 负责与LLM API进行交互。
- **主工作流 (`main` 函数):** `main` 函数是整个系统的“指挥中心”，它按照“检索 -> 构建 -> 生成 -> （可选）修正”的逻辑顺序，依次调用上述功能模块，完成对每一条测试

数据的翻译任务。

1.2 核心模块一：基于混合检索的语法知识注入

我们工作的核心在于如何让模型理解壮语复杂的语法结构。为此，我们设计并实现了一个精密的语法规则检索模块（`find_best_rule`函数），其核心是 **混合检索（Hybrid Retrieval）** 策略。该策略旨在结合传统检索方法的效率和现代语义检索的准确性。

- **关键词分数（Keyword Score）：**在 `calculate_keyword_scores` 函数中实现。这是一种高效的稀疏检索方法，通过计算待翻译句子与语法规则描述、示例及相关词汇之间的词语重叠度，快速定位在字面上最相关的规则。我们为不同部分的重叠赋予了不同权重，例如句子与例句中壮语词汇的重叠、以及与核心词汇表的重叠，以提升相关性判断的准确度。
- **语义分数（Semantic Score）：**在 `calculate_semantic_scores` 函数中实现。为了克服关键词匹配无法理解深层含义的局限，我们引入了语义向量检索。使用 `paraphrase-multilingual-MiniLM-L12-v2` 模型将所有语法规则的文本内容（描述与示例）编码为高维向量并缓存。在翻译时，我们同样计算待翻译句子的向量，并通过余弦相似度找到语义上最匹配的规则。
- **分数融合（Score Fusion）：**在 `find_best_rule` 函数中，我们将上述两种分数进行归一化，并通过一个可调节的权重参数 `hybrid_alpha` 进行线性融合：`final_score = α * keyword_score + (1-α) * semantic_score`。我们还引入了动态alpha策略，对于短句更侧重关键词匹配，对于长句更侧重语义理解，使得检索策略更具适应性。
- **(可选高级步骤：强模型重排（Re-ranking）)**：为了从召回的Top-N个候选规则中优中选优，我们的框架还支持一个由 `use_reranker` 开关控制的可选重排阶段。该阶段会调用一个更强大的语义模型（如`intfloat/e5-large-v2`），对这N个候选规则与原句的相关性进行更精细的二次打分，从而选出最精准的那条语法规则。

1.3 核心模块二：动态少样本与自我修正

除了语法知识，提供高质量的翻译范例和对结果进行迭代优化同样至关重要。

- **动态少样本检索 (Dynamic Few-shot Retrieval)**：在 `find_best_examples` 函数中，我们首先将语法书中所有的例句提取出来，构成一个“例句池”。接着，使用语义向量模型和余弦相似度计算，找出与待翻译句子在语义上最相近的K个例句（由 `num_few_shot` 参数控制）。这些动态检索出的范例能为LLM提供最直接、最相关的翻译风格和结构参考。

- **自我修正流程 (Self-Correction Loop)**： 为了追求极致的翻译质量，我们从多智能体（Multi-Agent）协作中获得灵感，设计了一个由 `use_self_correction` 开关控制的创新自我修正流程。该流程模拟了“翻译-审校”的两阶段工作流：
    i. **初步翻译:** 系统首先使用集成了所有知识的Prompt调用一次LLM，得到一个“初步翻译”。
    ii. **构建审校任务:** 接着，系统动态构建一个全新的审校Prompt，赋予LLM一个“顶级的壮语翻译审校专家”的新角色，并将【壮语原文】、【所有背景知识】以及【初步翻译】一同呈现给LLM。
    iii. **请求修正:** Prompt会明确指示LLM：
    > 请你严格根据背景知识，审视初步翻译是否存在任何不准确或不通顺的地方。如果存在错误，请直接输出你修正后的最终版本。

    iv. **获得最终翻译:** LLM在接收到这个审校任务后，会再次进行推理，输出一个质量更高、错误更少的“最终翻译”。

1.4 最终提示构建

我们最终的提示构建函数（`build_prompt`），它将以上所有检索到的知识模块化地、结构清晰地组织起来，其结构如下：

- **角色设定:** 首先，赋予LLM一个专家角色（“你是一个专业的语言学家和翻译家...”）。
- **知识注入:**
    - **相关词汇表:** 明确列出最相关的词汇及其翻译。
    - **翻译示例:** 提供动态检索到的、语义最相似的Few-shot范例。
    - **相关语法规则:** 呈现混合检索与重排后最优的语法规则描述和示例。
- **任务指令:**
    - **思维链引导 (Chain-of-Thought):** （可选）通过 `use_chain_of_thought` 开关，引导LLM先分析、再理解、最后生成，以激活其深层推理能力。
    - **明确输出格式:** 严格要求模型只输出最终的中文翻译。
- **待翻译句子:** 最后，附上待翻译的壮语句子。

2. 实验评估 (Experimental Evaluation)

我们在Kaggle平台上对Sub-task 1进行了评估，评估指标为NLTK实现的字符级BLEU分数。
最终取得了**0.57534**的score

3. 分析与讨论 (Analysis)

3.1 优势分析

- **混合检索的有效性:** 实践证明，单纯的关键词检索或语义检索都存在局限性。关键词方法快但可能不准，语义方法准但可能泛化过度。我们的混合检索策略通过加权融合，实现了优势互补，显著提升了语法规则检索的准确率和召回率。
- **自我修正的价值:** 自我修正流程在多个案例中成功修复了初步翻译中的语序错误、词汇误用等问题，证明了让模型“反思”其自身输出是一种行之有效的优化策略。
- **Prompt工程的重要性:** 精心设计的、结构化的Prompt，将不同来源的知识清晰地区分和呈现，对于引导LLM正确执行复杂任务至关重要。

3.2 错误分析与未来改进

尽管取得了不错的成绩，但模型在处理某些极其复杂的长句或包含歧义词汇的句子时，仍可能出现错误。

- **单一规则的局限性:** 当前系统每次只为句子注入一条“最佳”语法规则。然而某些复杂句可能同时涉及多条语法规则，单一规则不足以完全覆盖其句法结构。
- **修正的极限:** 自我修正流程的效果依赖于初步翻译的质量和背景知识的准确性。如果初步翻译错得离谱，或者检索到的知识本身就有偏差，修正流程也可能无力回天甚至引入新的错误。

基于以上分析，未来的改进方向可以包括：

- **更强的重排模型:** 引入一个更大、更专业的交叉编码器（Cross-Encoder）模型作为重排器，以进一步提升检索精度。
- **多轮自我修正:** 探索进行多于一轮的自我修正，看是否能持续提升翻译质量。

Sub-task 2

本次任务为 **困难且真实的低资源语言机器翻译任务 (Sub-task 2)**。与Sub-task 1相比，本任务场景更贴近现实，挑战也更大。具体要求为：在给定一本壮语（Zhuang）语法书、一本包含数万词条的壮汉词典以及数千句平行语料的条件下，将复杂的壮语句子准确地翻译成中文。

1. 实现方法

为应对Sub-task 2带来的开放词汇挑战，我们沿用并强化了为Sub-task 1设计的基于**检索增强生成（Retrieval-Augmented Generation, RAG）** 的高度模块化翻译框架。该框架通过为大语言模型（LLM）API（`qwen-max`）动态构建上下文极其丰富的提示（Prompt），在不进行模型微调的情况下，有效激发其在低资源语言上的翻译潜力。

1.1 系统与代码结构

为了保证实验的灵活性和可复现性，我们的所有实现都整合在了一系列独立的Python模块中，并通过一个主脚本`main.py`进行调度。

- **配置管理**: 我们使用`argparse`库进行参数管理，允许通过命令行开关灵活组合不同的策略，例如启用/禁用语法检索、设定相似例句的数量（`num_parallel_sent`）等。
- **核心模块**: 系统的关键功能被拆分为一系列独立的类和模块，每个模块职责单一，高度解耦：
    - `dictionary.py`：词典查询模块，负责处理词汇的精确匹配和模糊匹配，是解决开放词汇问题的关键。
    - `corpus.py`：平行语料检索模块，使用`BM25`算法检索与待翻译句子最相似的翻译范例。
    - `grammar.py`：语法知识检索模块，使用高级语义搜索从语法书中定位最相关的规则。
    - `prompt.py`：提示构建模块，负责将所有检索到的信息“装配”成最终的提示。
    - `model.py`：LLM交互模块，负责与云端API进行通信。
- **主工作流 (main函数)**: `main`函数作为系统的“指挥中心”，严格按照“检索 -> 构建 -> 生成”的逻辑顺序，依次调用上述功能模块，完成对每一条测试数据的翻译。

1.2 核心模块一：基于模糊匹配的词典知识注入

面对开放词汇的挑战，一个强大的词典查询模块是系统的基石。我们的`dictionary.py`实现了复杂的查询逻辑，远超简单的精确匹配：

- **精确匹配 (Exact Match)**：首先，系统会尝试对句子中的词进行精确查找。
- **模糊匹配 (Fuzzy Match)**：当遇到词典中不存在的词（未登录词）时，系统会启动一个三阶段的模糊匹配策略：

i. **最大正向/反向匹配**: 尝试识别输入词中是否包含词典中的某个词条。
ii. **编辑距离相似度**: 使用`rapidfuzz`库计算输入词与词典中所有词条的加权编辑距离（WRatio），召回拼写上最相似的候选词。

这个强大的模糊匹配机制使得我们的系统即使面对一些拼写错误或形态变化，也能大概率找到正确的词义解释，从而有效缓解未登录词问题。查询到的词义随后会被注入到Prompt中。

1.3 核心模块二：基于BM25的动态少样本检索

在词义不确定的情况下，提供高质量的翻译范例对于帮助LLM理解上下文和消解歧义至关重要。`corpus.py`模块实现了此功能：

- **BM25检索**: 该模块在初始化时，会使用`rank_bm25`库为所有平行语料建立一个高效的`BM25`索引。在翻译时，它会接收待翻译的句子，并利用该索引快速检索出在词汇重叠度上最相似的K个平行句对（由`num_parallel_sent`参数控制）。这些范例不仅展示了翻译风格，其上下文本身也为LLM推断未知词汇的含义提供了线索。

1.4 核心模块三：基于语义搜索的语法知识注入

我们工作的核心创新之一在于让模型理解壮语复杂的语法结构。为此，我们设计并实现了一个基于前沿技术的语法规则检索模块（`grammar.py`）：

- **语义向量化**: 该模块在初始化时，会使用一个强大的多语言句子嵌入模型（如`paraphrase-multilingual-mpnet-base-v2`）将语法书中所有规则的描述文本（`grammar_description`）编码为高维语义向量并缓存。
- **语义相似度计算**: 在翻译时，系统同样计算待翻译句子的语义向量，并通过余弦相似度在向量空间中找到与句子意思最相近的语法规则。这种方法超越了简单的字面匹配，能够理解句子的深层含义，从而定位到真正相关的语法知识。

1.5 最终提示构建 (Final Prompt Construction)

我们最终的提示构建函数（`prompt.py`中的`construct_prompt_za2zh`）是一个精密的“装配线”，它将以上所有检索到的知识模块化地、结构清晰地组织起来，其结构如下：

1. **角色设定**: 首先，赋予LLM一个专家角色（“你是一个专业的语言学家和翻译家...”）。
2. **知识注入**:
    - ***翻译示例 (Few-shot Examples)***: 提供由`corpus.py`检索到的、最相似的翻译范例。

    - ***词典解释 (Vocabulary Explanation)***: 提供由`dictionary.py`检索到的、关于句子中词汇的精确或模糊释义。
    - ***相关语法规则 (Relevant Grammar Rules)***: 呈现由`grammar.py`通过语义搜索找到的最优语法规则描述和示例。
3. **任务指令与待翻译句子**: 最后，给出明确的翻译指令和待翻译的壮语句子。

通过这种结构化的Prompt，我们为LLM提供了解决开放词汇翻译问题所需的全方位上下文信息，从而实现了高质量的低资源语言翻译。

2. 实验评估 (Experimental Evaluation)

我们在Kaggle平台上对Sub-task 2进行了评估，评估指标为NLTK实现的字符级BLEU分数。
最终取得了 **0.28303** 的成绩

实验结果清晰地表明，我们提出的基于语义搜索的语法注入模块，相比于仅依赖词典和例句的基线模型，在翻译质量上带来了显著的提升，证明了深层语法知识对于解决复杂翻译问题的重要性。

3. 分析与讨论 (Analysis)

3.1 优势分析

- **对开放词汇的鲁棒性**: 结合了精确匹配与多策略模糊匹配的词典模块，以及提供上下文线索的相似例句检索，共同构建了一个能有效应对未登录词和词义歧义的系统。
- **语义检索的优越性**: 语义语法检索能够超越字面限制，捕捉到句子和语法规则之间深层的联系，这是提升复杂句翻译质量的关键。
- **模块化设计的优越性**: 清晰的模块划分使得我们可以方便地对任一环节（如词典查询策略、语法检索算法）进行独立的实验和优化。

3.2 错误分析与未来改进

尽管取得了不错的成绩，但模型在处理某些极其复杂的长句或包含高度歧义词汇的句子时，仍可能出现错误。我们对一些错误案例（Bad Case）进行了分析，发现主要原因可能在于：

- **单一知识源的局限性**: 当前系统对三种知识源是独立检索后进行组合。更理想的方式可能是实现多知识源的协同推理，例如，用词典的查询结果来指导平行语料的检索。

- **LLM的内在幻觉**: 即使提供了所有正确的知识，LLM本身仍可能出现“幻觉”或不遵循指令的情况，导致生成意料之外的翻译。

基于以上分析，未来的改进方向可以包括：

- **引入重排（Re-ranking）机制**: 对BM25检索出的相似例句，使用更强大的语义模型进行二次排序，选出真正高质量的范例。
- **实现自我修正（Self-Correction）**：增加一个审校环节，让LLM对自己的初步翻译进行反思和修正。
- **探索多轮检索与推理**: 实现更复杂的Agentic流程，例如让模型先判断需要何种知识，再去主动检索，进行多轮迭代翻译。