**需要【去掉】的功能 ❌**

| **功能模块** | **具体功能** | **原因** | **影响** |
| --- | --- | --- | --- |
| **4.1 术语标记器** | 占位符替换（[TERM\_0]格式） | NLLB不识别，0%保留率 | 改为元数据记录位置 |
| **4.2 结构标注器** | 句型识别标记（[COND][ENUM]） | NLLB不识别结构标记 | 改为元数据记录 |
| **4.2 结构标注器** | 逻辑标记（因果、转折等） | 标记会被丢失 | 移到后处理推断 |
| **4.3 上下文增强器** | 指代消解（直接替换） | 可能破坏原文结构 | 改为标注不替换 |
| **1.3 元数据提取器** | 时代特征提取 | 测试中未体现价值 | 简化处理流程 |

**需要【调整】的功能 ⚠️**

| **功能模块** | **原功能** | **调整方案** | **调整原因** |
| --- | --- | --- | --- |
| **2.2 MT5语义编码器** | 复杂度评分 | 添加智能校正机制 | 简单句评估偏高 |
| **2.1 TibetanBERT分析器** | 敬语标记识别 | 只记录不标记 | 28%识别率需要改进 |
| **4.1 术语标记器** | 术语定位和替换 | 只定位不替换 | 保持原文完整性 |
| **5.1 参数优化器** | 重复惩罚设置 | 固定为1.25 | 测试得出的最优值 |
| **3.3 翻译记忆库查询** | 相似度计算阈值 | 调整为0.5 | 测试得出的最优值 |
| **2.3 上下文构建器** | 滑动窗口大小 | 固定为2-3句 | 避免除零错误 |

**需要【保留】的功能 ✅**

| **功能模块** | **功能** | **保留原因** | **优化建议** |
| --- | --- | --- | --- |
| **1.2 智能分段器** | 句子边界检测 | 85-90%准确率 | 继续优化边缘案例 |
| **1.1 文档结构分析器** | 文档类型判断 | 100%准确率 | 加强边缘案例处理 |
| **6.1 错误修正器** | 模式匹配修正 | 100%成功率 | 扩展规则库 |
| **6.1 错误修正器** | 重复消除 | 100%成功率 | 保持现状 |
| **5.3 缓存管理器** | 翻译缓存 | 91.8%命中率 | 使用large\_cache配置 |
| **3.1 术语库查询** | 精确匹配查询 | 66.7%覆盖率 | 扩充术语库 |

**需要【新增】的功能 ➕**

| **新增功能** | **所属模块** | **功能描述** | **添加原因** |
| --- | --- | --- | --- |
| **维度适配器** | 模型间接口 | TiBERT(768)→MT5(1024) | 解决维度不匹配 |
| **元数据管理器** | 预处理器 | 统一管理所有分析结果 | 替代标记方案 |
| **智能参数推荐** | 知识库 | 基于复杂度的参数库 | 优化翻译质量 |
| **后处理增强器** | 后处理模块 | 基于元数据的智能恢复 | 补偿NLLB不足 |

graph TB

A[藏文文档] --> B[文档预处理]

subgraph 预处理阶段

B --> B1[文档结构分析 ✅]

B --> B2[智能分段 ✅]

B --> B3[元数据提取 ⚠️简化]

end

B2 --> C[并行分析]

subgraph 分析阶段

C --> D1[TiBERT分析]

C --> D2[MT5分析]

D1 --> D1a[术语识别 ✅]

D1 --> D1b[敬语检测 ⚠️只记录]

D1 --> D1c[句法分析 ✅]

D2 --> D2a[复杂度评分 ⚠️+校正]

D2 --> D2b[语义编码 ✅]

D2 --> D2c[维度适配 ➕新增]

end

D1 --> E[元数据管理器 ➕新增]

D2 --> E

E --> F[知识库查询]

subgraph 知识库

F --> F1[术语库 ✅]

F --> F2[同义词库 ✅]

F --> F3[翻译记忆 ⚠️阈值0.5]

F --> F4[参数推荐 ➕新增]

end

B2 --> G[NLLB翻译]

F4 --> G

subgraph 翻译阶段

G --> G1[参数优化 ⚠️固定1.25]

G --> G2[批量翻译 ✅]

G --> G3[缓存管理 ✅large]

end

G --> H[智能后处理]

E --> H

subgraph 后处理阶段

H --> H1[错误修正 ✅扩展]

H --> H2[结构恢复 ➕增强]

H --> H3[术语恢复 ➕新增]

H --> H4[敬语增强 ➕新增]

end

H --> I[质量控制]

I --> J[最终输出]

**关键实现细节**

**4.1 元数据管理器（新增）**

Python

class MetadataManager:

def \_\_init\_\_(self):

self.metadata = {}

def add\_analysis(self, sentence\_id, analysis\_type, data):

"""统一管理所有分析结果"""

if sentence\_id not in self.metadata:

self.metadata[sentence\_id] = {}

self.metadata[sentence\_id][analysis\_type] = data

def get\_for\_translation(self, sentence\_id):

"""获取翻译所需的参数"""

meta = self.metadata.get(sentence\_id, {})

# 基于复杂度推荐参数

complexity = meta.get('mt5', {}).get('corrected\_score', 0.5)

return {

'num\_beams': 3 if complexity < 0.3 else 5 if complexity < 0.7 else 8,

'repetition\_penalty': 1.25, # 固定最优值

'temperature': 1.0 - complexity \* 0.2 # 动态调整

}

**4.2 后处理增强器（新增）**

Python

class EnhancedPostProcessor:

def process(self, translation, metadata):

"""基于元数据的智能后处理"""

# 1. 标准后处理

translation = self.error\_corrector.correct(translation)

translation = self.repetition\_remover.remove(translation)

# 2. 术语恢复（基于位置信息）

if 'terms' in metadata.get('tibert', {}):

translation = self.restore\_terms(translation, metadata['tibert']['terms'])

# 3. 敬语增强

if 'honorifics' in metadata.get('tibert', {}):

translation = self.enhance\_honorifics(translation, metadata['tibert']['honorifics'])

# 4. 结构优化

if 'sentence\_type' in metadata:

translation = self.optimize\_structure(translation, metadata['sentence\_type'])

return translation