您说得非常对！语言选择应该影响整个系统的处理流程，而不仅仅是标签显示。让我分析一下整个系统中需要根据语言调整的部分：

## 整个系统中需要语言调整的部分

### 1. \*\*视频转录阶段\*\*

- \*\*Whisper模型选择\*\*：不同语言可能需要不同的模型大小

- \*\*转录精度\*\*：某些语言可能需要更高的采样率

- \*\*时间戳精度\*\*：不同语言的语速差异

### 2. \*\*内容分析阶段\*\*

- \*\*LLM Prompt语言\*\*：分析prompt需要匹配目标语言

- \*\*内容类型识别\*\*：不同语言的内容结构可能不同

- \*\*知识点提取\*\*：语言特征影响概念识别

### 3. \*\*Summary生成阶段\*\*

- \*\*整合Prompt语言\*\*：生成summary的prompt需要匹配语言

- \*\*格式和结构\*\*：不同语言的summary格式可能不同

- \*\*专业术语\*\*：学科术语在不同语言中的表达

### 4. \*\*练习系统阶段\*\*

- \*\*题目匹配\*\*：根据语言匹配相应语言的题目

- \*\*AI助手对话\*\*：练习对话需要匹配用户语言

- \*\*提示和反馈\*\*：所有用户界面文本

### 5. \*\*用户界面阶段\*\*

- \*\*页面标题和按钮\*\*：界面文本本地化

- \*\*错误消息\*\*：系统提示信息

- \*\*帮助文档\*\*：用户指导文本

## 系统性的语言处理方案

### 方案A：全局语言配置系统

```

用户选择语言 → 全局配置对象 → 所有模块读取配置 → 动态调整处理逻辑

```

\*\*核心组件\*\*：

- `LanguageConfig` 类：管理全局语言设置

- `LanguageAwareProcessor`：所有处理器的基础类

- `PromptManager` 扩展：支持多语言prompt

### 方案B：语言感知的管道处理

```

视频输入 → 语言检测 → 语言特定处理 → 输出本地化

```

\*\*处理流程\*\*：

1. \*\*语言检测\*\*：自动或用户指定

2. \*\*处理器选择\*\*：根据语言选择合适处理器

3. \*\*Prompt本地化\*\*：动态加载对应语言prompt

4. \*\*输出格式化\*\*：按语言习惯格式化结果

### 方案C：模块化语言适配器

```

核心逻辑 + 语言适配器 → 多语言支持

```

\*\*架构设计\*\*：

- \*\*核心模块\*\*：语言无关的处理逻辑

- \*\*语言适配器\*\*：处理语言特定的部分

- \*\*配置驱动\*\*：通过配置文件管理语言差异

## 具体需要调整的模块

### 1. \*\*VideoProcessor\*\*

```python

# 当前：硬编码语言判断

if language.lower() == "english":

# 英文处理逻辑

# 建议：语言感知处理

class LanguageAwareVideoProcessor:

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

self.language = language

self.prompt\_manager = PromptManager(language)

self.transcription\_model = self.select\_model(language)

```

### 2. \*\*SummaryIntegrator\*\*

```python

# 当前：硬编码标签

segment\_label = f"### 分段 {i}"

# 建议：动态标签系统

class LanguageAwareSummaryIntegrator:

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

self.language = language

self.labels = self.load\_language\_labels(language)

```

### 3. \*\*PracticeLLMHandler\*\*

```python

# 当前：固定prompt

# 建议：语言特定prompt

class LanguageAwarePracticeHandler:

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

self.language = language

self.prompt\_template = self.load\_prompt(language)

```

### 4. \*\*QuestionMatcher\*\*

```python

# 当前：固定题目匹配

# 建议：语言感知匹配

class LanguageAwareQuestionMatcher:

def match\_questions(self, knowledge\_point, language):

# 根据语言匹配相应题库

```

## 推荐的整体架构

### 1. \*\*语言配置中心\*\*

```python

class LanguageConfig:

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

self.language = language

self.load\_config()

def get\_prompt(self, prompt\_type):

return self.prompts[prompt\_type]

def get\_labels(self, label\_type):

return self.labels[label\_type]

```

### 2. \*\*语言感知的处理器基类\*\*

```python

class LanguageAwareProcessor:

def \_\_init\_\_(self, language\_config):

self.language\_config = language\_config

self.language = language\_config.language

def process(self, data):

# 语言无关的核心逻辑

# + 语言特定的处理

```

### 3. \*\*统一的语言管理\*\*

```python

class LanguageManager:

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

self.supported\_languages = ['中文', 'english']

def create\_processor(self, processor\_type, language):

config = LanguageConfig(language)

return self.get\_processor\_class(processor\_type)(config)

```

这样的架构设计可以让整个系统真正支持多语言，而不仅仅是表面的标签翻译。您觉得这个方向如何？