项目位置：[https://github.com/NINEMINEsigma/202412-LLM-](https://github.com/NINEMINEsigma/202412-LLM-.git)

Lekit源码：<https://github.com/NINEMINEsigma/lekit>

**目 录**

[第1章 作品概述 1](#_Toc185453826)

[1.1 创意来源 1](#_Toc185453827)

[1.2 产生背景 1](#_Toc185453828)

[1.3 用户群体 1](#_Toc185453829)

[1.4 主要功能与特色 1](#_Toc185453830)

[1.5 应用价值 1](#_Toc185453831)

[1.6 推广前景 1](#_Toc185453832)

[第2章 问题分析 2](#_Toc185453833)

[2.1 问题来源 2](#_Toc185453834)

[2.2 现有解决方案 2](#_Toc185453835)

[2.3 本作品要解决的痛点问题 3](#_Toc185453836)

[2.4 解决问题的思路 3](#_Toc185453837)

[2.4.1 功能模块 3](#_Toc185453838)

[2.4.2 配置需求 4](#_Toc185453839)

[2.4.3 项目自测 4](#_Toc185453840)

[第3章 技术方案 5](#_Toc185453841)

[3.1 技术路线框架 5](#_Toc185453842)

[3.2 模块调用的简要说明 5](#_Toc185453843)

[3.3 低层工具模块与智能体实现 6](#_Toc185453844)

[3.3.1 基于vllm框架的LLM模型推理优化策略 6](#_Toc185453845)

[3.3.2 基于LangChain和Chat类型模型的智能体实现 7](#_Toc185453846)

[3.4 用例生成模块 7](#_Toc185453847)

[3.4.1 XX 7](#_Toc185453848)

[3.5 XX模块 8](#_Toc185453849)

[3.5.1 XX 8](#_Toc185453850)

[3.5.2 XX 8](#_Toc185453851)

[第4章 系统实现 8](#_Toc185453852)

[4.1 软件设计实现 8](#_Toc185453853)

[4.1.1 前端技术 8](#_Toc185453854)

[4.1.2 后端技术 8](#_Toc185453855)

[4.1.3 接口封装与调用 8](#_Toc185453856)

[4.1.4 系统集成 8](#_Toc185453857)

[4.2 用户界面 9](#_Toc185453858)

[4.2.1 主要使用界面 9](#_Toc185453859)

[4.3 数据来源 9](#_Toc185453860)

[4.4 数据训练 9](#_Toc185453861)

[4.5 改进过程 10](#_Toc185453862)

[4.5.1 XX 10](#_Toc185453863)

[4.5.2 XX系列模型的选择 10](#_Toc185453864)

[4.6 系统部署方法 10](#_Toc185453865)

[第5章 测试分析 11](#_Toc185453866)

[5.1.1 XX模块测试分析 11](#_Toc185453867)

[5.1.2 XX模块测试分析 12](#_Toc185453868)

[第6章 作品总结 12](#_Toc185453869)

[6.1 作品特色与创新点 12](#_Toc185453870)

[6.1.1 作品特色 12](#_Toc185453871)

[6.1.2 作品创新点 12](#_Toc185453872)

[6.2 应用推广 12](#_Toc185453873)

[6.3 作品展望 12](#_Toc185453874)

[参考文献 12](#_Toc185453875)

# 作品概述

## 创意来源

暂空。

## 产生背景

暂空。

## 用户群体

暂空。

## 主要功能与特色

主要功能：

暂空。

//可插图

特色：

1. **XXX：**暂空。

## 应用价值

简述暂空。

1. **XXX：**暂空。

## 推广前景

暂空。

# 问题分析

## 问题来源

暂空。

## 现有解决方案

暂空。

表 X XXX工具/产品/方法调研

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **工具/产品/方法** | **效率** | **成本** | **XX能力** | **XX内容** | **稳定性** | **生成模式** |
| 传统创作 | 低 | 高 | 极高 | 连续 | 无 | 固定 |
| XXX | 高 | 低 | 中 | 不连续 | 稳定 | 固定 |
| XXX | 高 | 低 | 中 | 连续 | 稳定 | 固定 |
| XXX | 高 | 低 | 中 | 连续 | 稳定 | 固定 |
| MyProject | 中 | 低 | 高 | 连续 | 较稳定 | 可定制 |

//可插图

**与XX相比本平台的优势：**

1. **效率与速度：**暂空。
2. **一致性与多样性：**暂空。
3. **节省成本：**暂空。

**与同类AI工具相比本平台的优势：**

1. **XX：**暂空。
2. **XX：**暂空。

**平台有待优化之处：**

1. **相较于同类AI工具XX的不足：**暂空。
2. **系统维护的稳定性：**暂空。

## 本作品要解决的痛点问题

1. 暂空。
2. 暂空。

## 解决问题的思路

### 功能模块

1. **低层工具模块：**lekit（lazy and easy kit），提供了项目中使用的绝大多数接口与工具，在使得代码更加易读且明确的情况下，还推动了项目耦合性的大幅降低与提高了极大程度的可维护以及扩展性。经由lekit封装的LangChain等核心内容，允许灵活地部署智能体并使用其服务，并通过定制的prompt及其重封装类的模板构造合适的上下文。  
   ***ProjectConfig：****该类型继承自lekit.Core.GlobalConfig，项目中的默认环境变量、资源文件管理、模块间通信等都由该类型负责，其使用将贯穿在后续的模块中*
2. **用例生成模块：**大模型将通过阅读给定的设计文档，生成可供人员阅读的测试用例文档或数据表格以及后续测试工作的自动路径AutoRuntimePath配置文件。  
   ***AutoRuntimePath****：包含了完整测试用例的信息，并提供了进一步工作的行为模板，自动化测试模块等模块可以通过该配置执行对用例的测试****输入：****目标软件的设计文档或说明书等具有完整软件使用规则与说明的docx文档****输出：****测试用例，AutoRuntimePath.json*
3. **自动化测试模块：**依据用例生成模块的产物，在目标平台上执行自动化测试。进一步结合智能体与提供的工具函数，自动化测试将在智能体对设计文档的理解下逐步对各个用例进行测试并通过网页反馈、视觉表观等方面得到与预期结果是否具有差异，并得出该用例的评测结果。该模块可以简要地测试软件的网页部分是否能够正常运行，暂定（并在运行过程中将允许交互的接口存入AutoRuntimePath中以供接口测试模块进一步测试）。  
   ***输入：****起始网址，AutoRuntimePath.json（可选）* ***输出：****核心页面的网页截图，操作过程的录制视频（可选），AutoRuntimePath.json（可选，若有输入的AutoRuntimePath.json则进行更新）*
4. **响应测试模块：**暂定（依据AutoRuntimePath对目标软件的接口进行接口测试与性能测试，并根据用户提供的数据文件，以数据驱动和大语言模型数据增加的方式对目标接口进行充分测试）
5. **单元测试模块：**暂空。
6. **集成测试模块：**暂空。
7. **（暂空）**

### 配置需求

1. **低层工具模块：**最低需要Python3.12，硬件需求依赖于用户选择的模型，除运行大模型与网络服务的部分外机器内存达到512MB即可。
2. **用例生成模块：**依赖于用户选择的模型。
3. **自动化测试模块与响应测试模块：**若目标软件不在本地，则需要网络

### 项目自测

* 未定

1. 数据格式：未定
2. 数据来源：未定
3. 数据获取方式：暂空
4. 数据特点：暂空
5. 数据规模：未定
6. 数据样例：

//可插图

1. 数据集用途：暂空

# 技术方案

## 技术路线框架

本项目的技术方案由高低两层技术架构组成。

第三方与底层API通过lekit部分聚合成多个核心，这些核心组成的低层架构将以更详细和统一的形式向高层技术架构提供具体实现而不需要考虑参数可能的类型与过程中存在的操作。

多个模块组成了高层技术架构，在这一层中的所有跨模块通信仅通过ProjectConfig类完成，所有的环境变量的初始化与动态配置也由此类完成，并且通过AutoRuntimePath对各个模块进行数据驱动，实现了代码中的低内聚，并且可以通过修改Config.json热更新而不需要停机

图 X 整体技术路线框架图

技术方案的具体层次结构和逻辑流向如下：

首先，暂空。

接着，暂空。

随后，暂空。

最后，暂空。

总结暂空。

## 模块调用的简要说明

本项目以软件包的形式对用户的需求提供支持，既可以通过各个模块内定义的Core类型手动控制各个模块的工作，使其能够独立完成任务，并以中间件的模式为二次开发提供实现。也可以依照设定的流程，结合ProjectConfig与AutoRuntimePath配置实现全流程的软件测试，以完整产品的模式直接提供服务。

## 低层工具模块与智能体实现

lekit包提供了诸如tool\_file、GlobalConfig等工具类型与工具函数来完成程序与本地资源的交互，并以统一的接口暴露给其他代码使用，并通过完整的OOP体系降低了二次开发的心智负担，并通过RAII技术减少了核心技术细节以外的琐碎代码；提供了诸如light\_llama\_core等核心类将llama.cpp、LangChain等LLM库重新封装以达到简化开发又能利用低级API的效率优势；提供了C-Like风格的工具与类型标注以便增强类型提示，使得开发者能够在不熟悉库中内容的情况下依然能通过IDE的智能提示快速上手lekit以及其引用的其他第三方库。

本项目在多种生成任务与测试任务中，其主要的输入数据为用户提供的设计文档与软件网页，低层工具模块主要职责是封装用户代码需要的对用户输入的深度解析和目标产物生成的功能。为此低层工具模块中集成的vllm+LLM（大语言模型）的深度学习能力与封装的Core模块（包括但不限于网络服务、多模态、动态反射等内容），能够很好的对高层技术架构提供帮助。

大语言模型 (LLM) 是一类基础模型，经过大量数据训练，使其能够像人类一样理解和生成文本以及其他形式的内容。这种模型有能力从环境中推断，生成连贯且与环境相关的响应，翻译、总结文本，执行各种任务。

项目使用的智能体采用了vllm + Qwen的技术方案，vllm 框架和 Qwen 模型在连环画生成平台的文本识别模块中分别扮演推理加速和核心处理的角色，前者优化模型推理效率，后者提供强大的语言理解和生成能力。

### 基于vllm框架的LLM模型推理优化策略

其中vllm 是一个基于剪枝技术的大模型推理加速工具，通过去除模型中的冗余参数，可以在保证模型性能的同时显著减少推理时间。vllm由于具备易于使用、具有最先进的服务吞吐量、高效的注意力键值内存管理（通过PagedAttention实现）、连续批处理输入请求、优化的CUDA内核等功能，成为本平台优化模型推理效率的选择。

首先vllm具备高度专业化的加载预训练模型的能力，且大多数主流模型都被vllm所支持，通过其提供的专门的API接口，可将预先训练好的大型语言模型加载到vllm框架中，简化了加载目标模型的过程，且确保了模型数据的完整性和准确性。

其次，vllm框架的模型剪枝功能具有高度的专业性和灵活性。剪枝是vllm的核心功能之一，它通过智能地删除模型中的冗余参数，降低模型复杂度，从而实现推理加速。在剪枝过程中，vllm提供了丰富的参数设置选项，根据文本理解模块具体需求和模型特点，可以通过调整这些参数来找到最佳的剪枝策略。

由于经过剪枝优化后的模型在计算过程中所需的内存和计算资源减少，因此推理速度得到了显著提升。同时，vllm还采用了先进的优化算法和技术，如PagedAttention、连续批处理和CUDA核心优化等，进一步提高了Qwen模型推理的效率和准确性。这些专业化的技术使得vllm在处理目标模型时能够保持高性能的同时，实现快速推理。

最后，vllm还支持分布式推理，这使得它能够在多台GPU上并行运行模型，进一步提高推理速度。这种分布式推理的支持使得vllm能够应对更大规模的模型和更复杂的推理任务，可以满足项目的用户端不断增长的需求。

### 基于LangChain和Chat类型模型的智能体实现

lekit具体应用了LangChain模型来实现文本理解、语义分析，工具函数调用，模板提示词链接等核心功能，并且使用的light\_llama\_core封装的ChatLlamaCppme提供聊天模型的支持，此类模型采用人类对齐技术进行微调，聊天模型拥有先进的工具使用和规划能力，可用于创建agent应用程序，具有很强的竞争力。

利用LangChain搭配Chat类型模型搭建的智能体，使用其强大的通用性和智能性来理解用户提供的文本、图片、页面内容，提取关键信息并在后续转化为进一步工作所需的结构化数据。

## 用例生成模块

暂空。

### XX

暂空。

## XX模块

暂空。

### XX

暂空。

### XX

暂空。

# 系统实现

## 软件设计实现

### 前端技术

暂空。

### 后端技术

暂空。

### 接口封装与调用

暂空。

### 系统集成

暂空。

## 用户界面

### 主要使用界面

暂空。

## 数据来源

暂空。

## 数据训练

为了全面、系统地评估项目的性能与效果，特制定以下训练测试方案：

1. 训练目标
2. **功能完整性：**暂空。
3. **适应性与稳定性：**暂空。
4. **用户体验：**暂空。
5. 训练方法与步骤
6. 单元测试

简述暂空。

评估指标如下：

1. 文本理解准确率：统计正确文本内容所占的比例。
2. 生成时间：记录单个产出的平均耗时。
3. 结果评分：根据人工对照进行主观评分。
4. 系统测试

简述暂空。

评估指标如下：

1. 系统稳定性：记录处理过程中出现错误、延迟或其他异常的次数与比例。
2. XX：未定。
3. 数据分析
4. **数据整理：**汇总单元测试、系统测试与用户体验测试的各项量化指标及主观评价。
5. **性能分析：**对比测试结果与预期目标，分析项目在功能完整性、艺术表现力、适应性与稳定性、用户体验方面的实际表现，识别优势与不足。
6. **结果解读：**结合具体测试案例，深入解读数据背后的原因。
7. **建议提出：**根据测试结果与分析，为项目的优化升级提供针对性改进建议，包括代码改进、参数调整等。

通过以上训练测试，在开发过程中定期对项目的综合性能进行全面、深入的评估，为其持续优化提供坚实的数据支持。

## 改进过程

### XX

简述暂空：

1. **实验设计：**暂空。
2. **数据收集：**暂空。
3. **数据处理：**暂空。
4. 测试结果：

暂空（XX从XX变为了XX…）。

### XX系列模型的选择

暂空。

## 系统部署方法

1. 系统需求
2. 硬件需求：

未定。

1. 网络环境：

需要网络 / 不需要网络。

1. 环境准备
2. 未定。
3. 暂空。
4. 部署

部署步骤：

1. 对项目的相关代码进行编译
2. 将服务端上传至服务器，并在指定或需求的目录下解压缩。
3. 部署XX模型到服务器机器的本地。
4. 根据首次启动的配置引导完成Config.json的配置
5. 服务启动与验证
6. 启动与测试：暂空。
7. 功能完整性验证：暂空。

# 测试分析

### XX模块测试分析

* 测试数据集

暂空。

* 测试过程

1. 数据预处理：暂空。

输出评估：未定。

* 测试结果

暂空。

* 分析与结论

暂空。

### XX模块测试分析

* 测试数据集

暂空。

# 作品总结

## 作品特色与创新点

### 作品特色

1. **XX：**暂空。
2. **XX：**暂空

### 作品创新点

1. **XX：**暂空

## 应用推广

1. **XX领域：**暂空

## 作品展望

//可插宣传图

暂空。

# 参考文献

[1]景奕昕.基于AI技术的软件测试系统研究[J].高科技与产业化,2024,30(11):39-40.

//更多引用