电网台区美化治理与打分系统 - 每日报告

报告日期: 2025 年 8 月 3 日
报告内容: 任务进度分析、改进建议、下一步计划

任务完成情况总览

任务编号	任务内容	完成状态	完成度	备注
1	GitHub 协作权限	【完成】已完成	100%	YilongJu 已被添加为协作者
2	Prompt 扩展至 100 行 +	【完成】已完成	100%	已达标，含丰富示例
3	打分系统结果输出	【完成】已完成	95%	系统已开发，待实际图片打分验证
4	End-to-end 代码实现	【未完成】未完成	25%	缺少核心治理功能，无端到端实现
5	多大模型对比实验	【部分完成】部分完成	60%	已有 OpenAI+Qwen，缺少 kimi/智谱
6	人员分工明细	【完成】已完成	100%	分工清晰明确
7	进度报告生成	【完成】本报告	100%	已生成详细评估

详细进度分析

【完成】任务 1: GitHub 协作权限设置

状态: 已完成 - 【完成】已完成: 所有代码文件已上传至 repository - 【完成】**Git** 提交记录: 最新提交”修改 prompt 模板, 将内容扩充到 100 行以上”- 【完成】协作者权限: YilongJu (juyilong@gmail.com) 已被添加为协作者 - 【完成】访问验证: 用户已确认可以访问 repository

【完成】任务 2: Prompt 优化扩展

状态: 基础完成 - 文件位置: `codespace/prompt.md` - 实际行数: 84 行核心内容（不含示例）- 基础要求: 【完成】已达标，含丰富示例 - ** 下一步重点 **: 把地

形因素坐标也加入 prompt，并且参考 `openaiApi.py` 分离 system prompt 和 user prompt

【完成】任务 **3:** 打分系统开发

状态: 系统已完成，需实际验证 - 后端实现: - 打分系统代码/`models.py`: 完整的数据模型 - 打分系统代码/`views.py`: 128 行完整 API 实现 - 支持评分配置管理、结果存储、分页查询 - 部署状态: - 登录地址: test.easysolar4u.com:8091 - 账号: mgxpjry / 密码: Bonck@123 - 待验证: 是否有实际台区图片的打分结果输出 - 建议: 团队演示一次完整的图片打分流程

【未完成】任务 **4: End-to-End** 代码实现

状态: 严重不足，缺少核心治理功能 - 【部分完成】现有组件（分散状态）: - `codespace/qwenMaxApi.py` (143 行): 千问 VL-Max API 调用脚本（已修复大括号问题，包含完整的错误处理）- `codespace/prompt.md` (84 行): 优质的 prompt 模板 - 美学评分函数：`PPXMGX/utils/beautify_config.py` (未集成) - 标注数据：治理前后对比数据已准备 - 【未完成】严重缺失的功能: - 核心治理引擎: 没有完整的大模型治理实现 - **End-to-End Pipeline**: 完全没有统一流程 - 打分集成: 治理和打分系统完全分离 - 可用的主程序: 没有可运行的完整脚本 - ** 实际状态**: - 治理功能：【未完成】缺失 - 打分系统：【完成】**Django web** 应用（独立）- 集成脚本：【未完成】不存在 - 紧急需求 **: 从零开始创建完整的 end-to-end 治理 + 打分脚本

【部分完成】任务 **5:** 多大模型对比实验

状态: 部分完成，需补充 - 已实现模型: -【完成】OpenAI o3 (已配置 API key 和环境变量) -【完成】千问 VL-Max-2025 (已配置百炼 API) - 计划中但未实现: -【未完成】Kimi K2 API -【未完成】智谱 GLM-4.5 API - 分工安排: 金建峰负责此项，需要补充 kimi 和智谱 GLM 的实现 - 建议: 添加智谱 GLM-4V，国内访问较稳定

【完成】任务 **6:** 人员分工明细

状态: 分工清晰完整 - 文件位置: 人员具体分工/人员分工明细.md - 团队结构: - 金建峰: Prompt 优化、数据准备、多模型对比、报告生成 - 樊文波: 打分系统开发、美观性评价、模型训练研究 - 季亢: 后端接口开发、模型训练研究 - 职责覆盖: 前端、后端、AI 模型、数据处理全方位覆盖 - 协作模式: 樊文波 + 季亢协作模型训练，分工明确

【完成】任务 **7:** 实施日志与进度报告

状态: 已建立完善的记录体系 - 日志文件: 每天实施日志/ 目录包含详细记录 - 最新进展 (2025-07-28): - 新增 150 个台区标注（总计 300 张图片）- 美观性评价工具开发完成 - 千问 API 研究和代码优化 - 报告质量: 本报告提供全面的任务评估

下一步行动计划

□ 类别一：完成未完成的核心任务

1. 多模态输入功能开发

- 图片输入支持: 图像预处理、base64 编码、多模态 API 调用
- 地形数据提取: 建筑物、道路、河流的矢量坐标
- 数据格式标准化: 统一空间数据的输入格式

2. End-to-End 核心功能开发 **(**任务 **4** 完成**)**

- 开发核心治理引擎: 创建完整的大模型治理引擎
- 构建统一 **Pipeline** 脚本: 整合治理 + 打分功能
- 集成美学评分函数: 将 `PPXMGX/utils/beautify_config.py` 集成到主流程

```
# 示例
import time
import wandb
from pathlib import Path
from beautification import beautification_pipeline
from evaluation import evaluation_pipeline

def call_beautification_function(image_path, image_data, model):
    """ 统一的美化治理 API 调用函数"""
    # 调用治理模型
    treatment_result = beautification_pipeline(image_path, image_data)

    # 计算美观性评分
    beauty_score = evaluation_pipeline(treatment_result)

    return {
        'input_data': image_data,
        'output_data': treatment_result['output_data'],
        'beauty_score': beauty_score,
        'input_tokens': treatment_result['input_tokens'],
        'output_tokens': treatment_result['output_tokens'],
    }
```

2. ** 多模型 **API** 完善 **** (**任务 **5** 完成**)**

- 实现 **Kimi K2 API** 接口
- 实现智谱 **GLM-4.5 API** 接口
- 统一模型调用接口

```
# 示例: unified_model_api.py
class ModelManager:
    def __init__(self):
```

```
        self.apis = {
            'openai': self._init_openai(),
            'qwen': self._init_qwen(),
            'kimi': self._init_kimi(),      # 新增
            'glm': self._init_glm()         # 新增
        }

    def _init_kimi(self):
        # Kimi K2 API 配置
        return KimiAPI(
            api_key=os.getenv('KIMI_API_KEY'),
            model_name='kimi-k2'
        )

    def _init_glm(self):
        # 智谱 GLM API 配置
        return GLMAPI(
            api_key=os.getenv('GLM_API_KEY'),
            model_name='glm-4v-plus'
        )

    ...
```

类别二：基于当前进度的扩展任务

1. ** 批量图像处理 Pipeline**

- 进度追踪: 实时显示处理进度
- 结果汇总: 批量结果统计和分析
- 多图并行处理: 支持批量台区图像治理

1 □ WandB 追踪集成

```
# 简化版 GIS 追踪器 - 重点追踪成本和效果
import wandb
import time

def log_beautification_results(prompt, input_data, output_data,
                               beauty_score, model_name, image_path,
                               input_tokens, output_tokens):
    """ 核心功能：追踪治理过程的关键指标"""

    # 各模型价格（每 1M tokens）- 自行修改
    pricing = {
        'qwen-vl-max': {'input': 0.1, 'output': 0.3}
    }
```

```
    cost = (input_tokens * pricing[model_name]['input'] / 1e6 +
            output_tokens * pricing[model_name]['output'] / 1e6)
    results_dict = {
        # 业务指标
        "beauty_score": beauty_score,
        "input_devices": len(input_data),
        "output_devices": len(output_data),
        "improvement": len(output_data) - len(input_data),

        # 成本指标（重要!）
        "input_tokens": input_tokens,
        "output_tokens": output_tokens,
        "total_cost_usd": cost,
        "cost_per_device": cost / len(input_data) if len(input_data) > 0 else 0,

        # 模型信息
        "model_name": model_name,
        "prompt_length": len(prompt),

        # 可视化
        "input_image": wandb.Image(image_path) if image_path else None,
    }

    # 记录所有关键数据
    wandb.log(results_dict)
    return results_dict
```

查看结果

访问 https://wandb.ai/你的用户名/gis-beautification 查看追踪结果

Artifacts 用途

- 保存完整的 prompt+ 输入输出数据,文档:https://docs.wandb.ai/guides/artifacts

2 □ 批量处理集成

```
# 批量处理示例
import multiprocessing as mp
from functools import partial

def process_single_image(task_data):
    """ 处理单张图像的工作函数"""
    image_path, image_data, model = task_data
```

```
    # 调用美化治理 API
    start_time = time.perf_counter()
    result = call_beautification_function(image_path, image_data, model)
    process_time = time.perf_counter() - start_time

    input_data = result['input_data']
    output_data = result['output_data']
    beauty_score = result['beauty_score']

    # 返回结果数据（不在子进程中调用 wandb）
    return {
        'image_path': str(image_path),
        'model': model,
        'input_data': input_data,
        'output_data': output_data,
        'beauty_score': beauty_score,
        'process_time': process_time,
        'input_tokens': result.get('input_tokens', 1500),
        'output_tokens': result.get('output_tokens', 800)
    }

def process_batch_with_tracking(input_list, models=['qwen-vl-max'], max_workers=4):
    """ 批量处理多张图像，使用 multiprocessing 并行"""

    # 初始化 WandB（主进程）
    wandb.init(project="gis-beautification", reinit=True)

    # 准备任务列表
    tasks = []
    for image_path, image_data in input_list[:5]:  # 限制 5 张图测试
        for model in models:
            tasks.append((image_path, image_data, model))

    # 使用进程池并行处理
    with mp.Pool(processes=max_workers) as pool:
        results = pool.map(process_single_image, tasks)

    # 在主进程中记录所有结果到 WandB
    for result in results:
        log_result = log_beautification_results(
            prompt=" 治理设备位置...",
            input_data=result['input_data'],
            output_data=result['output_data'],
            beauty_score=result['beauty_score'],
            model_name=result['model'],
```

```
            image_path=result['image_path'],
            input_tokens=result['input_tokens'],
            output_tokens=result['output_tokens']
        )

    # 记录批量处理汇总
    wandb.log({
        'batch_images_count': len(set(r['image_path'] for r in results)),
        'batch_avg_beauty_score': sum(r['beauty_score'] for r in results) / len(results)
    })

    wandb.finish()
    return results
```