图片清洗提效文档

> 核心目的是：在图片进入标注前，先用 Dify 工作流完成批量初筛，减少低质量图片进入标注流程带来的人工判断成本和数据分发不均。

1. 基础信息

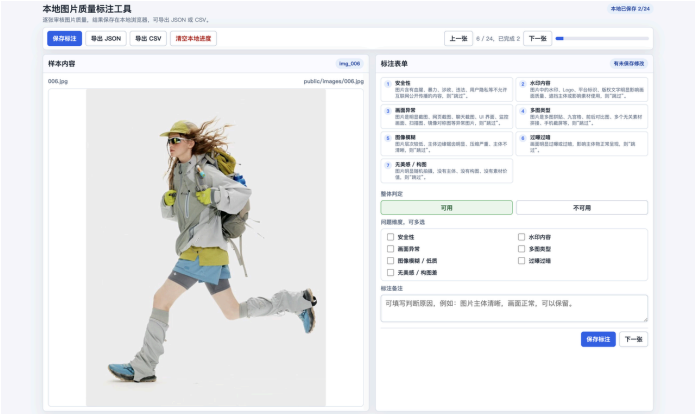
在图片理解和图片标注任务中，低质量图片会带来两个问题：

- 一是标注同学拿到的数据难度不一致，容易造成标注积极性和效率差异，
- 二是人工需要反复加载、查看、判断，重复消耗人力。

所以本次工作流要解决的是：在进入标注前先筛掉明显低质图，让后续标注拿到更稳定、更均衡的数据。

问题	影响	处理目标
图片质量参差不齐	不同标注同学任务难度不一致	先筛掉明显低质图
人工逐张判断	耗时、重复、消耗人力	用工作流做批量初筛

标注界面（本地）：



评测指标：

- **误杀率**：低于5%
- **效率指标**：数据质量判断完全机标

最终目标：

全量上线后，图片质量判断工作不再由标注同学来做。只用人工抽检不合格区和每日专人判断灰度区。

2. 方案设计

这个方案的关键是分工：脚本负责客观、可量化的问题；LLM 负责需要理解图片内容的问题，灰度区暂存当前工作流无法清晰处理的风格图。

脚本负责	LLM 负责
图片能否访问	是否有明显水印或版权字
尺寸是否过小	是否是截图、拼图、网页图
是否过曝、过暗、低对比	主体是否明确，是否有素材价值
是否存在可量化的明显模糊	是否适合进入后续标注

这样做有四个好处：明显低质图不再消耗 LLM；LLM 能拿到脚本检测结果作为参考；边界图片进入灰度区等待人工复核；后续问题定位更清楚。

2.1 脚本迭代策略及方法

本次图片预筛采用“规则约束 + 多指标判断 + 异常分流”的组合策略，核心思路如下：

- **前置分流**：先过滤图片小尺寸、极端比例等无效样本，减少无效数据进入后续流程。
- **质量判断增强**：不再只看全图 `blur_score`，而是结合亮度、过曝/过暗比例、对比度、中心区域清晰度、缩放后清晰度综合判断。
- **本次解决方式**：新增中心区域指标、高频纹理密度、小连通区域数量、边缘密度等特征，识别密集水印/文字覆盖带来的“虚假清晰”。
- **边界控制**：脚本只处理客观低质问题，内容语义、审美、安全风险继续交给 LLM 判断。

> 脚本层定位为前置客观质检，用于优先过滤尺寸、曝光、模糊、低对比等稳定可判定的低质图片，减少无效样本进入 LLM 审核；同时将脚本检测结果作为客观参考写入 User Prompt，辅助模型判断，降低幻觉和主观误判。

2.2 提示词搭建策略及方法

本次 Prompt Engineering采用“规则定义 + 模型约束 + Badcase 迭代”的搭建思路，核心工作如下：

- **审核维度拆解**：将图片清洗任务拆分为安全性、水印内容、画面异常、多图类型、图像模糊、过曝过暗、无美感/构图等维度，并明确每个维度的“通过 / 跳过”判断边界。

- **Prompt 规则约束**：通过提示词约束模型只判断脚本无法稳定处理的主观问题，例如水印、截图、拼图、主体不明确和素材价值，避免模型重复判断分辨率、曝光等客观指标。
- **输出格式标准化**：设计固定 JSON 输出结构，要求模型输出 `decision`、`confidence`、`reason`、`problem_dimensions`；其中 `decision` 支持 `normal`、`skip`、`review` 三类，用于承接合格、不合格和灰度区。
- **Badcase 驱动优化**：基于实际测试中出现的误判、漏判 case，持续调整判断标准和 Prompt 表达方式，使模型在水印、低质图、截图、多图拼贴等场景下判断更稳定。

> 本次提示词设计的重点，是把图片清洗任务拆成明确审核维度，并约束模型的判断边界和输出格式。再结合 Badcase 持续优化，让 LLM 审核结果更稳定、可解析、可复查。

2.3 主要卡点及解决

脚本漏判问题修正：脚本漏判是本次迭代中最典型的问题之一。部分主体发白、发虚的低质图，因为水印、小字、纹理边缘较多，导致 `blur_score` 被抬高而误判通过。下面按主要卡点分别说明。

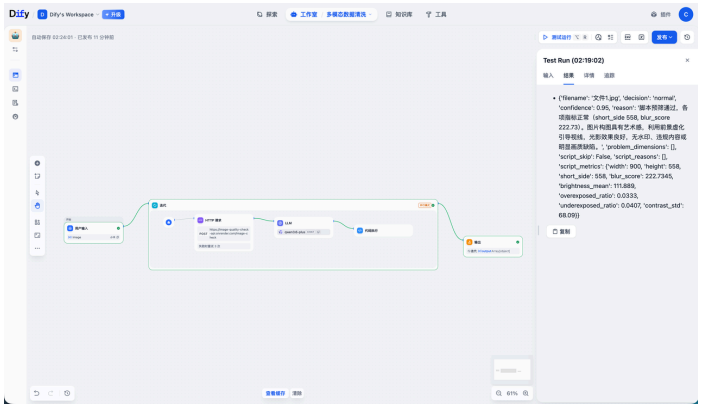
卡点 1：URL 无法直接进入视觉模型，图片类型不匹配

- **问题**：CSV 中输入的是图片 URL，但视觉模型实际需要读取图片文件；如果直接把 URL 传给 LLM，模型只能看到文本，无法稳定完成图片审核。
- **解决**：在 LLM 前新增“下载图片”节点，通过 HTTP GET 将 URL 转换为 Dify 可识别的 File 类型，再将 `下载图片 / files` 作为视觉输入，保证模型真正读取图片内容。
- 工作流示例：

LLM 前增加 HTTP 请求节点

卡点 2：LLM 承担过多判断，成本高且不稳定

- **问题**：如果所有图片都直接进入 LLM，尺寸过小、过曝、过暗、低对比等明显客观问题也会消耗模型调用，成本高且判断不够稳定。
- **解决**：新增后端脚本作为前置客观质检，先检测分辨率、清晰度、亮度、曝光、对比度等可量化指标。明显低质图直接 `script_skip=true`，不再进入 LLM；通过预筛的图片再由 LLM 处理水印、截图、主体价值等主观问题。



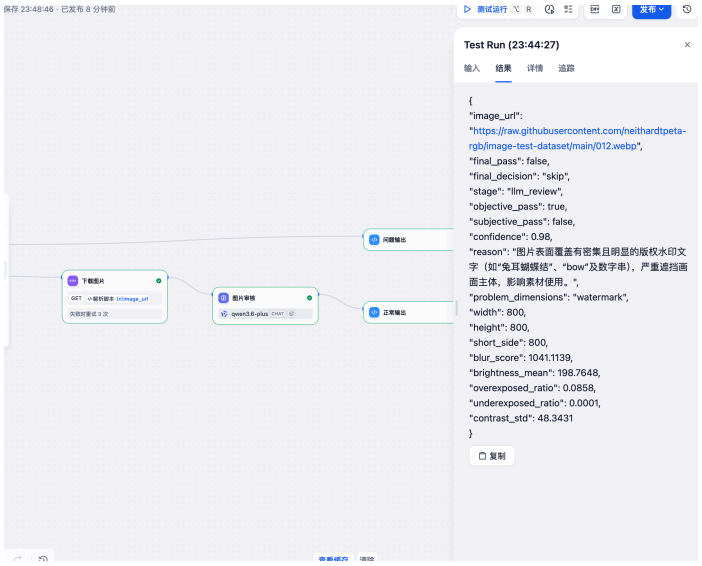
只用 LLM的工作流展示

```python
from typing import Any
from urllib.parse import urlparse

import httpx
import numpy as np
from fastapi import FastAPI, HTTPException, Request
from PIL import Image, UnidentifiedImageError
from starlette.datastructures import UploadFile


ALLOWED_EXTENSIONS = {"jpg", "jpeg", "png", "webp"}
ALLOWED_IMAGE_FORMATS = {"jpeg", "png", "webp"}
MAX_UPLOAD_BYTES = 20 * 1024 * 1024
MAX_URL_IMAGE_BYTES = 10 * 1024 * 1024
IMAGE_URL_TIMEOUT_SECONDS = 10.0

CENTER_CROP_RATIO = 0.6
RESIZED_BLUR_MAX_SIDE = 512
HEURISTIC_ANALYSIS_MAX_SIDE = 512

MIN_SHORT_SIDE = 384
MIN_IMAGE_AREA = 200_000
MAX_ASPECT_RATIO = 4.0
MIN_ASPECT_RATIO = 0.25

OVEREXPOSED_BRIGHTNESS_MIN = 185.0
OVEREXPOSED_RATIO_MIN = 0.08
WASHED_OUT_BRIGHTNESS_MIN = 190.0
WASHED_OUT_CONTRAST_MAX = 55.0
SEVERE_WASHED_OUT_BRIGHTNESS_MIN = 200.0
SEVERE_WASHED_OUT_CONTRAST_MAX = 65.0
UNDEREXPOSED_BRIGHTNESS_MAX = 45.0
UNDEREXPOSED_RATIO_MIN = 0.35
```

部分代码维度和阈值展示

卡点 3：脚本漏判主体发白、发虚的低质图

- **问题**：上一版脚本主要依赖 `blur_score` 判断图片清晰度，但部分图片虽然主体已经发白、发虚、低质，画面表面却存在大量水印、小字和纹理边缘，导致 `blur_score` 被异常抬高，脚本误判为清晰图片并放行，最终增加了 LLM 审核压力。
- **解决**：通过 badcase 复盘，将这类问题从“单一模糊检测”改为“发白、低对比、假清晰、密集水印覆盖”的组合判断。后端脚本新增亮度、过曝比例、对比度、主体区域清晰度等规则，用于提前拦截这类表面锐利但主体质量差的图片，减少低质样本进入后续 LLM 节点。
- case 示例：



明显问题的图片仍然进入 LLM 节点

问题图片

但现在有一个问题：某些图片主体严重发白、发虚、低质，但因为表面覆盖了大量锐利水印、小字、纹理，导致 blur_score 很高，脚本误判为通过，最终进入 LLM。

例如这类图：

* 主体发白、发雾；
* 画面有大量密集水印或文字纹理；
* 主体识别质量很差；
* 但 blur_score 因为小字和水印边缘被抬高。

请你修改后端脚本，不要改变接口字段名称，不要破坏 Dify 当前解析逻辑。仍然返回：

{
"script_skip": true 或 false,
"script_reasons": [],
"metrics": {...}
}

请增加更完整的客观预筛规则。目标是：能用脚本稳定判断的低质图提前 script_skip=true；不能稳定判断的语义问题继续留给 LLM。

描述问题的思路展示

卡点 4：模型输出不稳定，影响批量统计和复查

- **问题**：LLM 容易输出自然语言、Markdown 或不固定字段，导致后续结果难以解析，也不方便批量统计图片是否可用及具体原因。
- **解决**：通过 Prompt 约束模型只输出固定 JSON 字段，包括 `decision`、`confidence`、`reason`、`problem_dimensions`，并在输出节点统一转换为 `final_pass`、`final_decision`、`stage` 等字段，保证结果可解析、可追踪、可复查。

System Prompt 示例（精简版）：

📝 **核心约束**：LLM 只处理脚本无法稳定判断的主观问题，不重复判断尺寸、曝光、分辨率等客观指标。

- 输入：图片 URL、脚本预筛原因、关键质量指标。
- 判断范围：安全风险、水印/Logo、截图或 UI、多图拼贴、主体不明确、明显素材价值不足。
- 输出要求：只返回 JSON，不输出 Markdown、代码块或额外解释。
- 固定字段：`decision`、`confidence`、`reason`、`problem_dimensions`；`decision=review` 时进入灰度区。

输出示例

```
{
  "decision": "normal",
```

```
    "confidence": 0.95,
    "reason": "图片主体明确，内容正常，适合作为素材保留",
    "problem_dimensions": []
}
```

卡点 5：URL 与“一行多个 URL”的尝试

在解决 URL 图片下载问题后，又尝试过“一行多个 URL”的批量输入方式。这个问题的核心不是 URL 能不能被下载，而是批量输入形态是否稳定。

当一行里包含多个 URL 时，工作流需要先把多个 URL 归一，再逐个做脚本检测、条件分支、LLM 审核和结果汇总。

这个方向能跑通，但链路更长，运行中更容易报错；一旦失败，也更难定位到底是哪一个 URL 出问题。因此这个版本最后没有作为主方案保留。

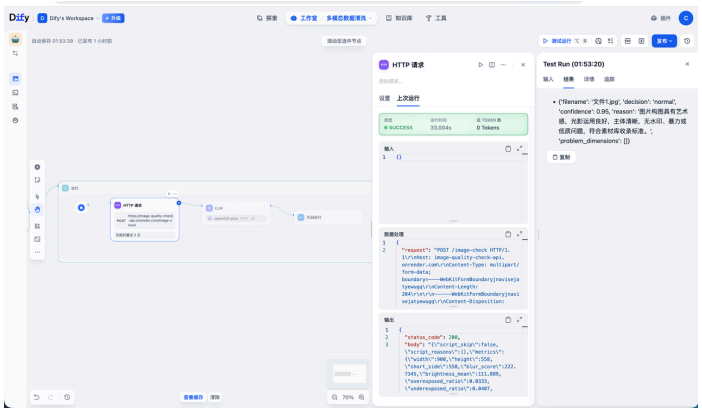


图 2：通过 HTTP 请求处理 URL 图片

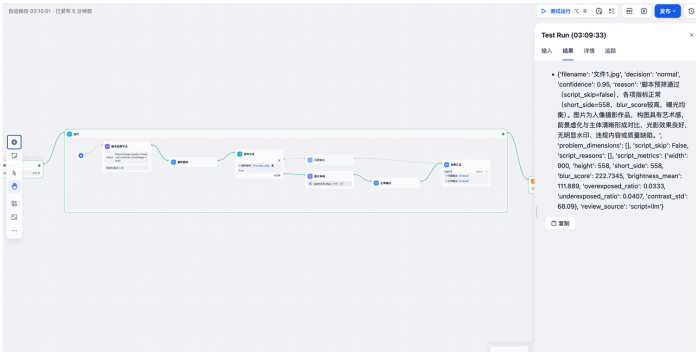


图 3：URL 图片进入脚本预筛与 LLM 审核的尝试链路

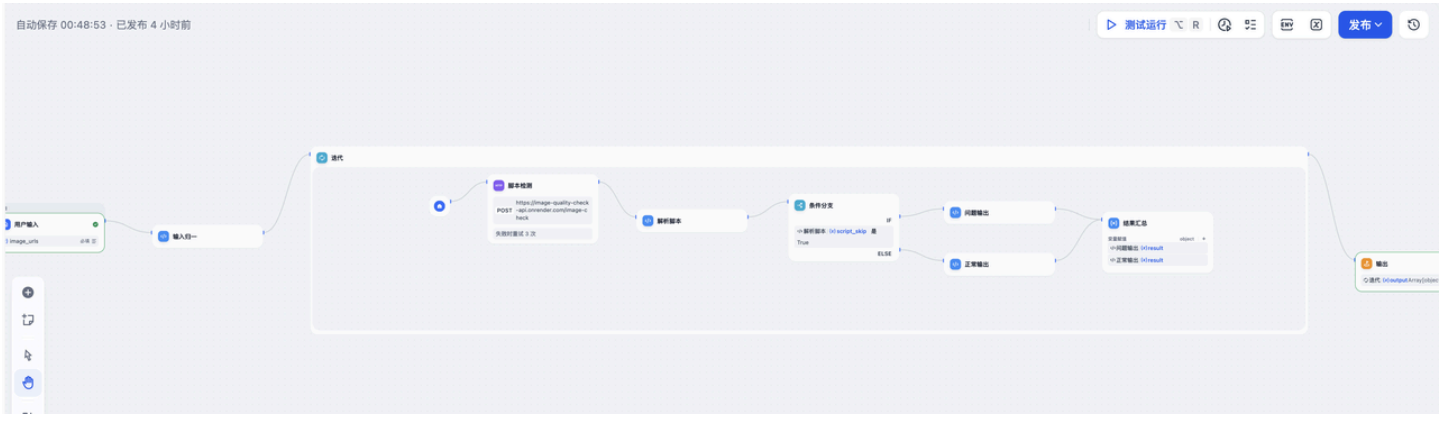
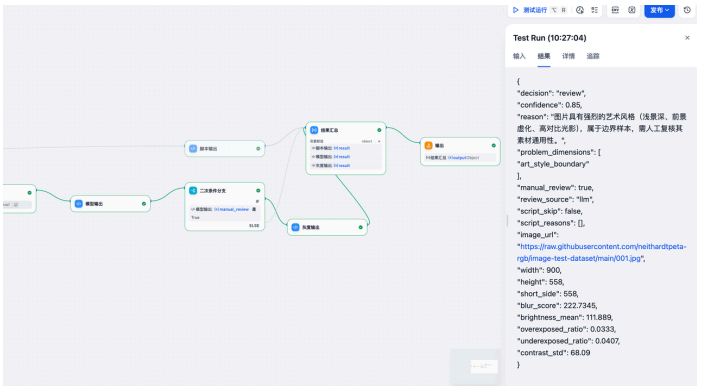


图 4：一行多个 URL 的尝试版本，链路更复杂，稳定性较低

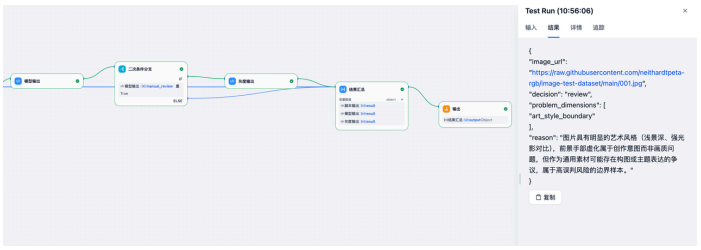
卡点 6：二值判断导致漏放率过高

新增灰度区后，原来的结果不再只是“合格 / 不合格”二分。模型遇到难以明确判断的图片时，如果直接放行，会把边界图混入合格数据；如果直接判为不合格，又可能误杀仍有价值的素材。因此需要在模型输出后增加一个灰度区，作为边界样本的暂存池。

问题	影响	处理方式
只有合格 / 不合格两个出口	边界图片容易被误放行或误拦截。	新增灰度输出，专门承接难以辨别的图片。
新增分支后字段容易变多	批量 CSV 后续不好统计，也不利于复查。	在结果汇总里统一问题输出、模型输出和灰度输出。
人工复核没有优先级	审核精力仍然会平均分散。	把灰度区作为人工优先复核池，先处理最不确定的样本。



冗余案例展示



优化后案例展示

📌 **最终处理规则**：脚本明确判定为低质的图片，进入问题输出；LLM 明确判断可用或不可用的图片，进入模型输出；LLM 无法明确判断、需要人工再看一眼的图片，进入灰度输出。灰度区不是“不合格”，而是人工复核前的暂存状态。

2.4 最终方案及其优势

最终方案收敛为：**CSV 一行一个 URL，Dify 逐行批量运行**；脚本先做客观质量预筛，明显不合格的图片直接输出问题原因；通过预筛的图片再进入 LLM 审核。模型输出后新增二次分支，把结果拆成**问题输出、模型输出、灰度输出**三类，最后统一汇总。

最终流程

```
CSV 图片 URL
↓
脚本先判断客观质量
├─ 明显不合格：进入问题输出
└─ 客观通过：下载图片，交给 LLM 审核
↓
LLM 输出结果
├─ 判断明确：进入模型输出
└─ 难以辨别：进入灰度输出，等待人工复核
↓
```

```
11    结果汇总：问题输出 / 模型输出 / 灰度输出
```

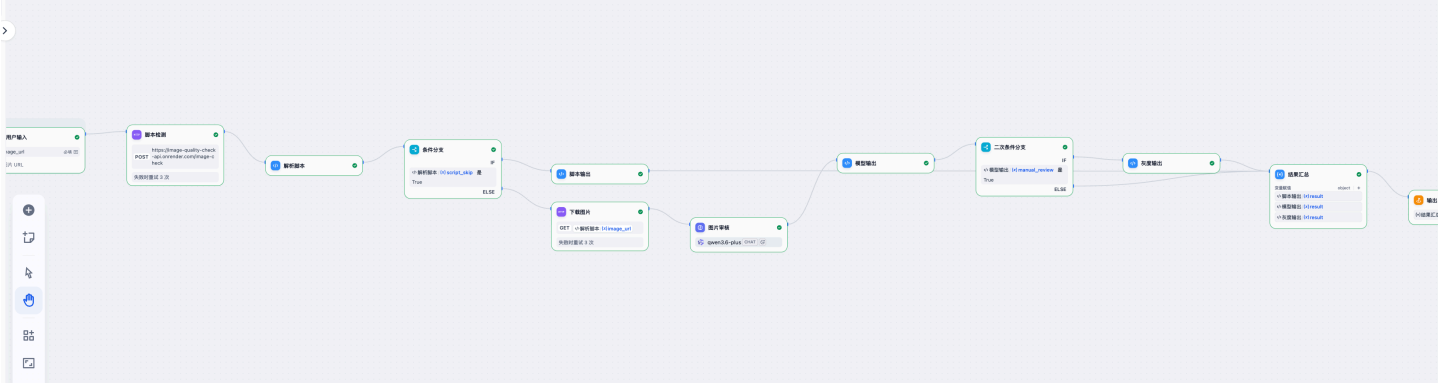
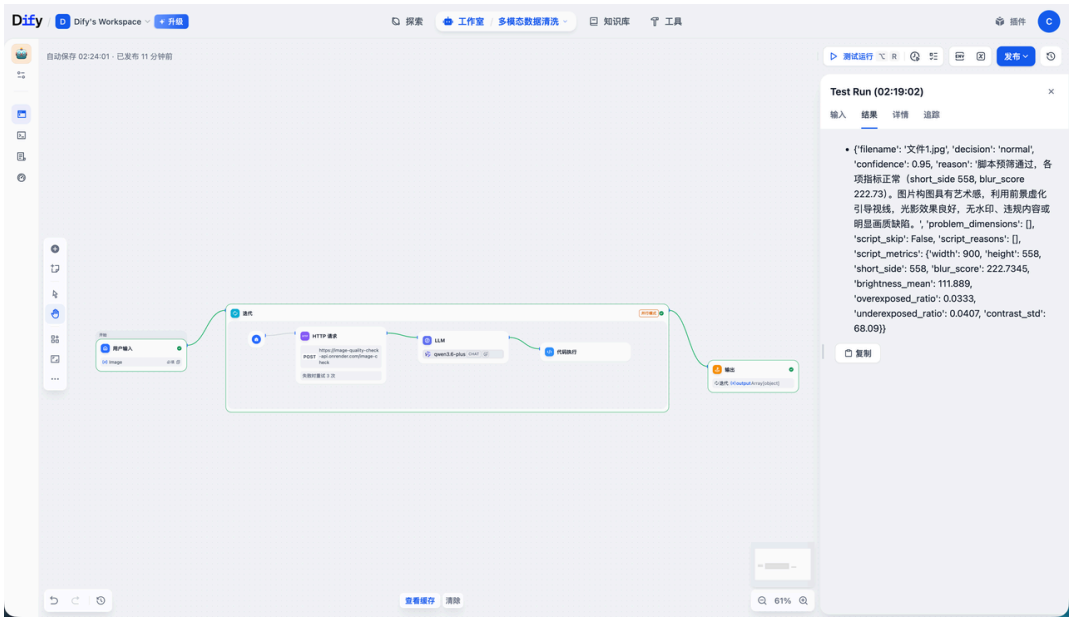


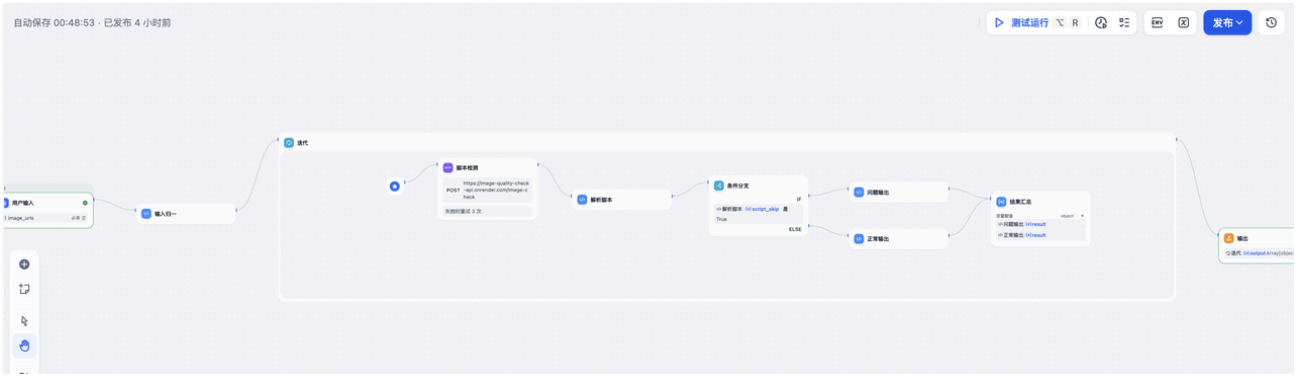
图 5：最终版，新增灰度区后的批量检测工作流

3. 迭代记录和实验日志

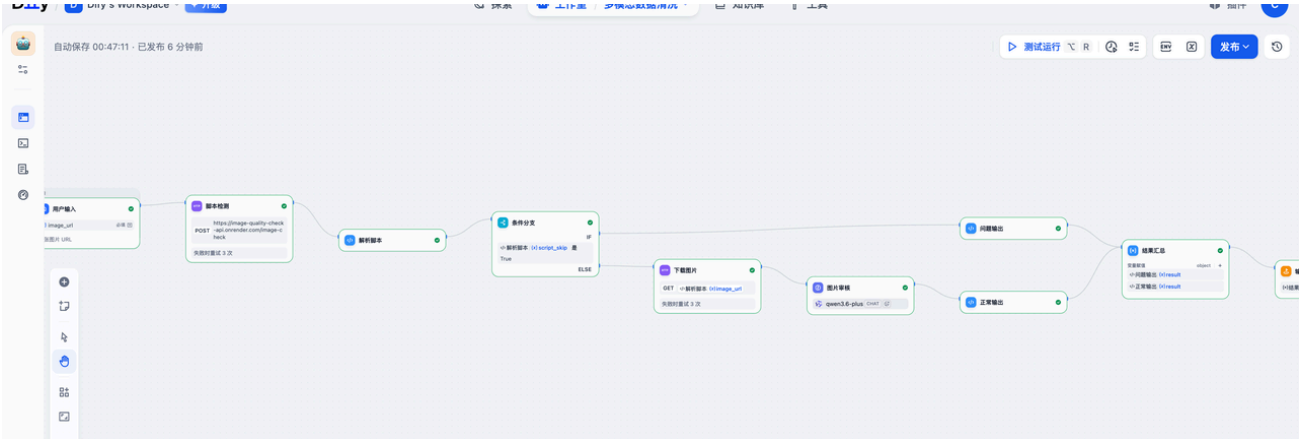
1. 第一版：手动上传图片，最多 10 张。能验证方向，但不适合大批量。
2. 第二版：加入 URL 图片处理，通过 HTTP 请求把 URL 图片下载给模型看。



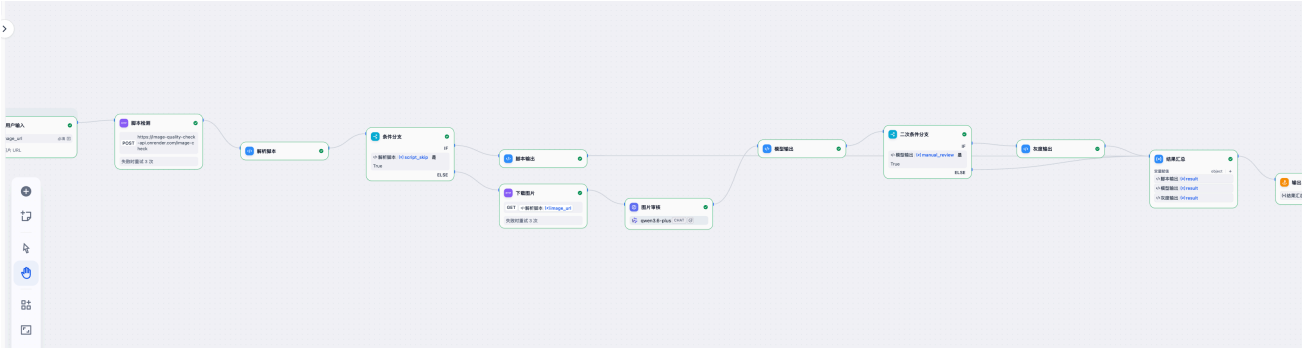
3. 第三版：加入脚本预筛，先拦截尺寸、曝光、模糊等客观问题。
4. 第四版：尝试一行多个 URL，但稳定性不够，定位问题困难。



5. 第五版：改成 CSV 一行一个 URL，Dify 批量逐行运行。



6. 最终版：在第五版基础上新增灰度区，用来暂存模型难以明确判断、需要人工复核的边界图片。



当前节点图已经增加灰度输出。灰度区用于暂存模型难以一次判断的边界图片；在人工复核完成前，不直接计入合格或不合格。上线后按同一比例估算，灰度区约 0.6 万张，用来观察边界样本占比。

全量上线后，清洗工作流累计处理图片数据约 9 万张。不合格图片约 3 万张，其中脚本预筛拦截约 1.7 万张，LLM 审核拦截约 1.3 万张；灰度区约 0.6 万张。

指标	数量	说明
累计处理图片	约 8 万张	真实上线后的累计处理规模
合格图片	约 4.4 万张	可进入后续标注或素材使用
灰度区图片	约 0.6 万张	进入人工复核
不合格图片	约 3 万张	被脚本或 LLM 明确拦截
脚本预筛拦截	约 1.7 万张	主要是尺寸、曝光、模糊、低对比等客观质量问题
LLM 审核拦截	约 1.3 万张	主要是水印、截图、主体价值等内容问题

这个结果说明，清洗工作流已经从离线测试进入真实生产使用。脚本预筛承担了约 1.7 万张明确低质图的提前拦截，减少了 LLM 的无效审核；LLM 继续处理脚本难以判断的内容问题，并额外通过灰度区承接边界样本。

4. 本地辅助工具的沉淀

为了减少重复操作，额外做了三个本地小工具。

工具	作用
GitHub 图片 CSV 生成器	把 GitHub Public 仓库里的图片整理成 CSV
图片 URL 转换工具	把一行多个 URL 拆成一行一个 URL
Dify 结果 CSV 分流工具	把合格、不合格和灰度区图片拆开，并区分脚本拦截、LLM 拦截和人工复核池

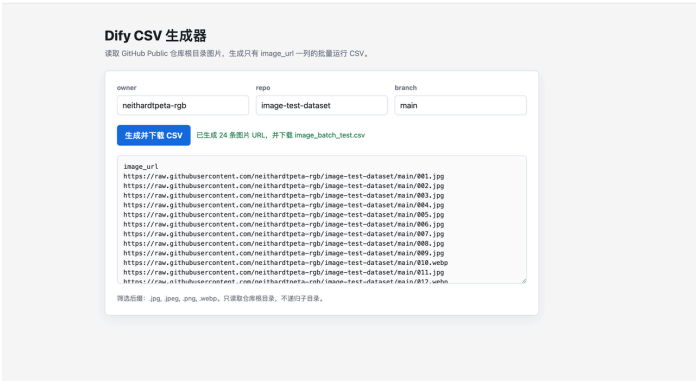


图 5：GitHub 图片 CSV 生成器

图 6：把一行多个 URL 转换成一行一个 URL

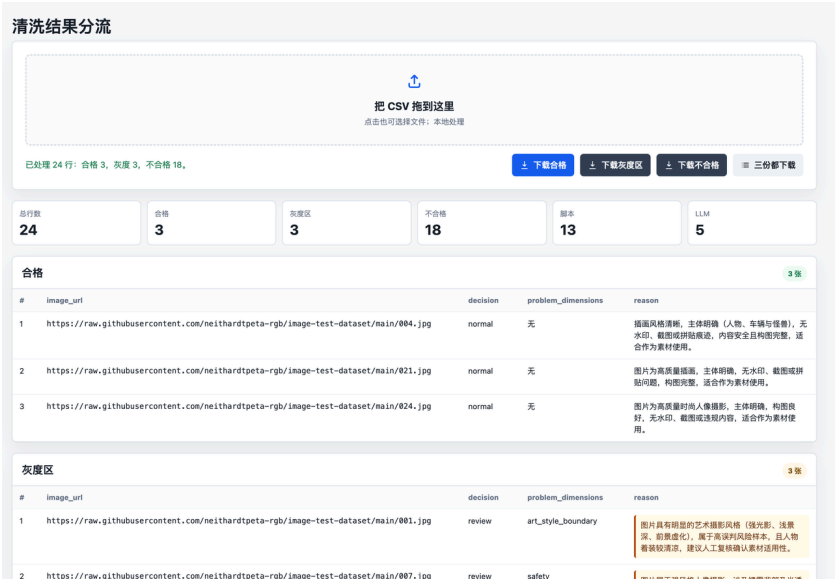


图 7：清洗结果分流，拆分合格、灰度区和不合格图片

结果分流的价值在于把清洗结果拆成三个后续处理池：**合格图片**可以进入后续使用；**灰度区图片**优先人工复核，用来沉淀边界判断标准；**不合格图片**回看具体原因，再判断是调整 Prompt、脚本阈值，还是补充人工规则。这样后续调优不会只看总量，而是能直接定位问题出在哪一类样本上。

5. 最终收益总结

收益	说明
图片质量判断无需标注同学	**只需每日处理灰度区和定期少样本抽查，标注人效17/h 提升至 21/h， 每条节约 40 秒。**
处理规模扩大	上线后累计处理约 8 万张图片，已经从小样本验证进入真实生产流程。
降低 LLM 审核压力	脚本预筛拦截约 1.7 万张客观低质图，这部分不再需要交给 LLM 判断。
减少低质图进入标注	累计识别不合格图片约 3 万张，减少明显低质图继续分发给标注同学。
方便复查调优	结果区分脚本拦截、LLM 拦截和灰度区，后续可以分别调整脚本阈值、Prompt 规则和人工判断标准。

6. 方法论分享

1. 提效的核心在于**把原有 SOP 拆得足够细致**，然后对每个流程通过对模型能力的边界和脚本能力边界做判断。
2. 先把输入标准化（做数据的前置处理）。CSV 一行一个 URL，比在工作流里兼容复杂输入更稳定。
3. 不要让 LLM 包办所有判断。客观问题交给脚本，主观内容交给 LLM。
4. 输出不要只看合格或不合格，还要保留拦截来源和灰度区，方便后续调优。
5. 难以辨别的图片不要强行二分。单独设灰度区，比把边界样本混进合格或不合格更稳。
6. 搭工作流的顺序建议是：先跑通，再稳定，再统一输出。

7. 后续优化方向

1. 持续抽检机判不合格区，重点观察误杀率是否因为数据样态变化而上升。
2. 把灰度区样本作为人工复核重点，定期沉淀边界判断标准，减少长期依赖人工经验判断。
3. 按拦截来源拆分调优：脚本误判优先调整阈值和规则，LLM 误判优先调整 System Prompt 或 User Prompt。
4. 定期复盘 8 万张上线数据后的新增 Badcase，更新脚本、Prompt 和结果分流工具的统计数据。
5. 如果图片描述阶段频繁出现因数据生产样态变动的风格图片被误杀，明显低质图片被漏放等问题，需要反向更新清洗规则、脚本阈值和灰度区标准。

(注：内容由 AI 生成，请谨慎参考)