# DeepEyes Eval 流程介绍

基于强化学习 (RL) 的多模态代理项目

尹超

https://github.com/Visual-Agent/DeepEyes

# Contents

# DeepEyes 项目介绍

DeepEyes 构建在 **VeRL** (Volcano Engine Reinforcement Learning) 框架之上。

- 核心目标: 训练多模态代理（Agent），使其能够在推理链中直接整合视觉信息。
- 能力: 主动调用 `image_zoom_in_tool` 查看细节，无需依赖冷启动或监督微调。

**核心机制**

端到端强化学习 (End-to-End RL): 利用结果奖励信号引导模型涌现出视觉搜索、比较等能力。 激励模型 “通过图像思考” **(Thinking with Images)**。

# 评估流程介绍 (Evaluation)

DeepEyes 评估采用 **两步法 (Two-step Approach)**:

1. 推理 (Inference): 模型生成回答 (含 Thinking & Tool Call)。
2. 打分 (Scoring): 使用 LLM-as-a-judge (通常为 Qwen-2.5-72B)。

## 环境准备

先拉取官方 docker image 后，进入 workspace，再确保下载了相关模型和数据集（例如推理模型下载命令如下）：

```
HF_ENDPOINT=https://hf-mirror.com huggingface-cli download
ChenShawn/DeepEyes-7B \
--local-dir ~/models/DeepEyes-7B \
--resume-download
```

评估脚本通过 OpenAI-compatible API (vLLM) 通信。需启动 两个 服务：

1. Target Model (被评估模型)

```
vllm serve /path/to/DeepEyes-7B-v1 \
    --port 8000 \
    --trust-remote-code \
    --gpu-memory-utilization 0.9
```

2. Judge Model (打分模型)

```
vllm serve /path/to/Qwen-2.5-72B-Instruct \
    --port 18901 \
    --served-model-name "judge" \
    --tensor-parallel-size 4
```

# V* 评估 - Step 1: 推理 (Inference)



```
python eval/eval_vstar.py \
  --model_name "DeepEyes-7B-v1" \
  --api_key "EMPTY" \
  --api_url "http://localhost:8000/v1" \
  --vstar_bench_path "/path/to/vstar" \
  --save_path "./results" \
  --eval_model_name "default" \
  --num_workers 8
```

参数解释:

- model_name: 输出文件夹名，用于区分实验。
- api_key: 鉴权密钥，本地一般用“EMPTY”。
- api_url: 被评测模型 API 地址。
- vstar_bench_path: V* 数据集本地路径。
- save_path: 结果保存根目录。
- eval_model_name: 请求体中的 model 字段。
- num_workers: 并发推理线程数。

# V* 评估 - Step 2: 打分 (Scoring)



```
python eval/judge_result.py \
  --model_name "DeepEyes-7B-v1" \
  --api_key "EMPTY" \
  --api_url "http://localhost:18901/v1" \
  --save_path "./results" \
  --eval_model_name "judge" \
  --num_workers 8
```

参数解释:

- `model_name`: 须与 Step 1 一致，定位中间结果。
- `api_key`: 鉴权密钥。
- `api_url`: 打分模型 (Judge) API 地址。
- `save_path`: 结果读取目录。
- `eval_model_name`: Judge 模型在 vLLM 的服务名。
- `num_workers`: 并发打分线程数。

# 结果解读 (Output Interpretation)

- 中间结果: `save_path/model_name/result_*.jsonl`
  - 包含: Prompts, Raw Model Outputs, Parsed Answers.
- 最终分数: `save_path/model_name/final_acc.json`
  - 生成于打分步骤之后。
  - 包含: 各类别准确率及总体准确率。

Usage Tip: 确保 `save_path` 和 `model_name` 在两步中完全一致，否则打分脚本无法找到推理结果。

# 数据结构分析

# 输入数据结构 (Input Data)

训练数据通常为 **Parquet** 格式，每条数据代表一个训练样本。

- `data_source`: 数据来源标识 (e.g., `"rag_v2-train"`).
- `prompt`: 对话历史 (Role: `system`, `user`).
- **`env_name`**: 关键字段。指定工具环境 (e.g., `"visual_toolbox_v2"`).
- `reward_model`: 奖励计算配置 (e.g., `ground_truth`).
- `images`: 图像数据（路径或二进制）。

参考: verl/workers/agent/envs/visual_agent/generate_trainset.py

# 输入数据示例 (Input JSON)




```
{
  "data_source": "deepeyes-train",
  "prompt": [
    { "role": "system", "content": "You are a helpful assistant..." },
    { "role": "user", "content": "What is the color of the flower?..." }
  ],
  "env_name": "visual_toolbox_v2",
  "ability": "qa",
  "reward_model": {
    "style": "rule",
    "ground_truth": "The color is white."
  },
  "extra_info": { "id": "sa_4988", "answer": "The color is white." }
}
```

# 输出数据结构 (Output Data)

结果保存为 JSONL 文件，每行一个 JSON 对象。

- `image`: 输入图像文件名
- `question`: 输入问题
- `answer`: 标准答案 (Ground Truth)
- `status`: 执行状态
- **`pred_ans`**: 模型预测并提取的简短答案
- **`pred_output`**: 完整的对话交互记录
  - 包含 思维链 `<think>`
  - 包含 工具调用 `<tool_call>`

参考: `results/deepeyes-7b/result_direct_attributes_deepeyes-7b.jsonl`

# 输出数据示例 (Output JSON)




```
{
  "image": "sa_4988.jpg",
  "question": "What is the color of the flower?",
  "answer": "The color of the flower is white.",
  "pred_ans": "white",
  "pred_output": [
    {
      "role": "assistant",
      "content": "<think>\nI need to check the details of the flower center.\n</
think>\n<tool_call>\n{\"name\": \"image_zoom_in_tool\", \"arguments\": {...}}\n</
tool_call>\n<answer>The color of the flower is white.</answer>"
    }
  ],
  "status": "success"
}
```

# 打分机制详解 (Scoring Mechanism)

# 总体策略 (Overall Strategy)

采用“规则匹配优先 + LLM 裁判兜底”的混合评价策略，兼顾效率与语义灵活性。

## 打分流程

对每个测试样本按序执行：

1. 预处理: 清理模型输出（如去除 `\boxed{}`）。
2. 规则匹配 (Rule-Based): 字符串精确匹配。命中则直接得分，不调用 LLM。
3. LLM 裁判 (LLM-as-a-Judge): 若规则未命中，构造 Prompt 发送给裁判模型（Qwen-2.5-72B），判断语义一致性。

# 规则匹配 (Rule-Based Check)

优先检查模型输出是否符合标准格式，节省 API 成本。

- 单字符匹配: 输出单个字符 (如 “A”) 且不冲突。
- 带点选项匹配: 输出包含点 (如 “A.”) 且包含正确选项。
- 包含匹配: 模型输出完整包含标准答案字符串 (e.g., “A. The towel is blue”)。

# LLM 裁判 (LLM-as-a-Judge)

当输出为自然语言或格式不规范时触发。使用 Few-Shot Prompt 指导裁判。

- Prompt 构造:
  - 指令: 判断 `[Model_answer]` 与 `[Standard Answer]` 语义是否一致。
  - 示例 **(ICE)**: 内置 7 个示例 (肯定/否定/颜色/方位等)，明确判定标准。
- 判定逻辑:
  - 若裁判返回 `Judgement: 1` -> 1.0 分。
  - 否则 -> 0.0 分。

# 评测集差异 (V* vs HRBench)



| 特性 | **V* Benchmark** | **HRBench** |
| --- | --- | --- |
| 标准答案 | 强制设为 'A' + 文本 | 读取真实选项字母 |
| 规则匹配 | 强依赖 'A' 字母 | 动态比较选项字母 |
| Prompt | "A. [Answer]" 格式 | 原始带选项文本 |

# 示例场景 (Scenarios)

假设标准答案: **`"A. The apple is red"`**

1. 输出 “A”:
   -> 规则匹配 (准确) -> **1.0 分**
2. 输出 “A. The apple is red”:
   -> 规则匹配 (完全包含) -> **1.0 分**
3. 输出 “The apple is clearly red”:
   -> 规则失败 -> LLM 判定语义一致 -> **1.0 分**
4. 输出 “It is green”:
   -> 规则失败 -> LLM 判定不一致 -> **0.0 分**

# 奖励机制详解 (Reward Mechanism)

# 奖励函数组成 (Reward Components)

基于 `verl/utils/reward_score/vl_agent.py`，奖励函数由三部分组成：

1. Accuracy Reward: 答案是否正确。
2. Formatting Reward: 格式标签是否规范。
3. Conditional Tool Bonus: 关键 - 激励“用工具做对题”。

**设计哲学**

通过混合奖励信号，引导模型不仅仅是“猜对”答案，而是建立正确的“视觉思考”路径。

1. Accuracy Reward
- 依赖 LLM Judge 判定。
- Code:

```
if '1' in response:
    acc_reward = 1.0
elif '0' in response:
    acc_reward = 0.0
```

2. Formatting Reward
- 检查 XML 标签闭合 (<think>, <answer>, <vision>)。
- Code:

```
# 标签不成对 -> -1.0 惩罚
if count_think_1 != count_think_2:
    is_format_error = True
format_reward = -1.0 if error else
0.0
```

# 条件工具奖励 (Conditional Tool Bonus)

这是激励 “Thinking with Images” 的核心机制。

**触发条件**

只有当 模型使用了工具 (vision > 0) 且 答案正确 (acc > 0.5) 时，才能获得奖励。

```
# 只有 "用了工具" 且 "答对了"，才能拿到这 1.0 的 tool_reward
tool_reward = 1.0 if count_vision_1 > 0 and acc_reward > 0.5 else 0.0
```

防止模型为了拿奖励而乱用工具（无效调用）或只用工具不答题。

# 最终奖励公式 (Final Formula)



加权求和得出最终 score:

$$R_{\text{total}} = 0.8 \times R_{\text{acc}} + 0.2 \times R_{\text{fmt}} + 1.2 \times R_{\text{tool}}$$

- 权重分析:
  - $R_{\text{tool}}$ (1.2) > $R_{\text{acc}}$ (0.8)。
  - 强力引导：模型“用工具做对”比“纯盲猜做对”收益更高。
  - $R_{\text{fmt}}$ (0.2) 配合 −1.0 的惩罚项，确保基本格式合规。