预训练 LLM 评估：工程实践者手册

三本头研究报告 · Part: engineer

本报告截稿日期: 2026-05-25　　共 8 章

端到端最小路径

读者已写过 eval pipeline——本章只画一条最小调用链作锚点，后续章节挂细节。一次完整 eval 的物料流是六步：

1. **ckpt 落盘** → `model_step_120000/` 含 `config.json`、`tokenizer.json`、shard 化的 `model-*.safetensors`、可选 `generation_config.json`。
2. **加载** 到 vLLM / HF / SGLang backend，绑定 tokenizer。
3. **Task 物化**：YAML 描述的 task 拉数据集、渲染 prompt、构造 request 列表。
4. **Batched inference**：按 request type（log-likelihood / generate-until）分桶批跑。
5. **Metric 聚合**：每条 request 的 raw output 喂 filter / scorer，task 级 reduce 出 acc / pass@k / EM。
6. **Artifact 产出**：结果 JSON + per-instance log + 上传到 leaderboard / W&B / Google Sheet。

跑一遍 7B base ckpt 的 MMLU+HellaSwag+GSM8K+ARC+HumanEval 在 8×A100 80G 上约 35–50 分钟，瓶颈在 GSM8K 的 generate-until 段。

一条真实命令

```
lm_eval \
  --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B,tensor_parallel_size=8,dtype=b
  --tasks mmlu,hellaswag,arc_challenge,gsm8k,humaneval \
  --num_fewshot 5 \
  --batch_size auto \
  --output_path results/llama31-8b/step120000 \
  --log_samples \
  --apply_chat_template false \
  --seed 1234
```

`--log_samples` 强烈建议默认开——它把每条样本的 prompt / target / logprob / 模型 raw output 落到 `samples_<task>.jsonl`，是后续 audit、contamination 检查、与他人数字对账的唯一可信源。无 `--log_samples` 的 leaderboard 数字基本不可复现。

1 / 64

`--apply_chat_template false` 对 base ckpt 是必须的——一旦误开，tokenizer 会塞进 `<|im_start|>system\n` 之类 chat template，base 模型从未见过，分数会暴跌 5–20 pp。这是新人最常见的 bug 之一。

中间产物与可观察性

`--log_samples` 之外，工程角度还有几个产物需要存档：

- **`stderr` / `stdout`**：harness 自身的 progress bar、warning（如 dataset version 拉取、tokenizer fallback）会写 stderr。生产环境用 `2>&1 | tee` 双写，防止 OOM 或 CUDA error 时 log 丢失。
- **GPU 监控**：跑全套 5 task 时用 `nvidia-smi dmon -s u -i 0-7 -o T > gpumon.log &` 后台跑；HellaSwag 等 MCQ 任务在 vLLM 上 utilization 90%+，GSM8K generate-until 段在 stop sequence 触发后 utilization 会跌到 30–40%——这是发现 stop sequence 配错的间接信号。
- **Wall-clock 表**：`time` 命令包装，按 task 分摊。如果 MMLU 跑了 40 min（正常 12–18 min），多半是某个 subset 拉数据集时网络重试。

关键工程产物

跑完后输出目录如下：

```
results/llama31-8b/step120000/
├── meta-llama__Llama-3.1-8B/
│   ├── results_2026-05-25T12-30-21.json   # 顶层 acc / acc_norm / stderr
│   ├── samples_mmlu_abstract_algebra_2026-05-25T12-30-21.jsonl
│   ├── samples_hellaswag_2026-05-25T12-30-21.jsonl
│   ├── samples_gsm8k_2026-05-25T12-30-21.jsonl
│   └── ...
└── llama31-8b.log                          # stdout 全量 log
```

`results_*.json` 的顶层字段：

```
{
  "results": {
    "mmlu": {"acc,none": 0.6841, "acc_stderr,none": 0.0038},
    "mmlu_abstract_algebra": {"acc,none": 0.43, "acc_stderr,none": 0.0497},
    "hellaswag": {"acc,none": 0.6112, "acc_norm,none": 0.7920},
    "gsm8k": {"exact_match,strict-match": 0.5072, "exact_match,flexible-extract"
    "humaneval": {"pass@1,create_test": 0.348}
  },
  "config": {"model": "vllm", "model_args": "...", "num_fewshot": 5, "batch_size
  "versions": {"mmlu": 1.0, "hellaswag": 1.0, "gsm8k": 3.0},
  "git_hash": "8f4e2d1",
  "date": "2026-05-25T12:30:21Z"
}
```


`versions` 字段决定数字是否可比——同 task 跨 `Version` 不保证一致（lm-eval-harness 在 `task.py` 里允许 task author bump version 后修改 prompt template，旧数字与新数字不能直接比）。

每一步都有坑，本部分如何展开

下面列出每一步最容易踩的工程坑，对应章节链接到后续展开：

- **ckpt 加载阶段**：tokenizer 与 base model 错配（如复用 chat-tuned tokenizer 跑 base 评测）、`pad_token_id` 未设导致 batched inference 输出全 EOS、`max_position_embeddings` 与 task `target_length` 不匹配。→ 第 2 章 lm-eval-harness 内部 `_loglikelihood_tokens` 详解。
- **Task 渲染**：few-shot 示例的随机种子、prompt 末尾换行符、`doc_to_text` / `doc_to_target` 的 token 边界。→ 第 2 章。
- **Backend 差异**：vLLM 与 HF Transformers 的 logprob 数值不一致（vLLM 在 logits 处理上使用 chunked prefill，与 HF 在第 16 位以下有差异）。Biderman 等的 2024 lm-eval-harness 主 paper 明示 HF 为 reference，vLLM 数字与 HF 在 MMLU 上偶有 0.2–0.5 pp 抖动[^biderman2024lessons]。→ 第 3 章框架对比。
- **Stop tokens / EOT**：GSM8K 生成时如果 stop sequence 配错，模型会一路生成到 `max_tokens=512`，pass@1 数字虚低 5–10 pp；HumanEval 没正确截到 `def` 边界会把测试代码也生成出来，sandbox 执行报错被判 fail。→ 第 4 章经典 benchmark 实操。
- **Reasoning era 的 max_tokens**：DeepSeek R1 / o3-mini 等 reasoning 模型 thinking token 平均 8k–32k，常规 `max_tokens=2048` 直接截断在 thinking 中间。→ 第 5 章。
- **Submission JSON**：传统 leaderboard 要求 `results_*.json` 直接上传；OpenEvals hub 退役后散落到 200+ 垂直 leaderboard，每家 schema 不同[^hf-leaderboard-retirement]。→ 第 3 章。

一个 pretrain run 的 eval cadence

7B–32B scale pretrain 项目典型评测节奏：

3 / 64

| Frequency | 跑什么 | Wall-clock | 工具 | |---|---|---|---| | 每 1k step | 训练 loss / val perplexity | 秒级 | 训练框架自带 | | 每 5k step | 5–8 task 的 lightweight probe（in-training-probing） | ~3 min | probe inference[^liu2026probing] | | 每 20k step | OLMES Base Easy（10 task 子集） | ~20 min | `olmes` CLI | | 每 100k step | OLMES Base Main + DCLM 53-task core | ~2 h | `olmes` + `dclm` | | Final / milestone | OLMES 43 task 全套 + HELM Capabilities + custom held-out | ~6 h | OLMES + HELM | | Release | 上述全部 + leaderboard 提交 + 第三方 audit | days | 全栈 |

后两节细化中间层（probe / Base Easy / Base Main）的工具链与 cadence 选择，本章不再展开。要点是：**不要把 final-milestone 那套全跑频率拉到每 5k step**——单次 OLMES 43 task 跑完一次的 GPU-hour 等于一个 lightweight probe 跑 200 次。in-training-probing paper 实测每 checkpoint 1h → 3min 的 20x 加速[^liu2026probing]，是中间监控的关键。

本部分阅读路径

剩余章节按"工具栈深度 → benchmark 类别"两条线展开：

- **第 2 章** 深读 lm-evaluation-harness 源码结构（`lm_eval/api/`、`lm_eval/tasks/`、`lm_eval/models/`）与 task YAML 写法。读者跑过 harness 但想自己加 task 或 debug request 路径，从这章入手。
- **第 3 章** 横向对比 OLMES / lighteval / HELM v2 / DCLM——为什么需要四个框架、各自的 scale-out 模型、什么场景选什么。
- **第 4–7 章** 按 benchmark 大类（经典 / reasoning / live / 长上下文+多模态+agent）给逐 benchmark 的命令、prompt、log 片段、与 paper 数字对齐的 debugging checklist。

Part II（调研）讨论 benchmark 自身的 critique（contamination、validity、saturation），Part III（决策）给出"什么 stage 跑什么 benchmark"的总图。本 Part I 只解决"跑得起来、跑得对、数字可复现"三件事。

引用

[^biderman2024lessons]: Biderman, S. et al. (2024). *Lessons from the Trenches on Reproducible Evaluation of Language Models*. arXiv:2405.14782.

[^hf-leaderboard-retirement]: clefourrier @ HuggingFace. (2025-03-13). *It's been a wild ride, folks :) (end of the Open LLM Leaderboard)*. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard/discussions/1135.

[^liu2026probing]: Liu, Z., Lun, T., Wen, Z. et al. (2026). *Fast and Accurate Probing of In-Training LLMs' Downstream Performances*. arXiv:2604.01025.

4 / 64

为什么要读源码

EleutherAI 的 lm-evaluation-harness（PyPI `lm-eval`，命令行 `lm_eval`）是 HF Open LLM Leaderboard v1/v2 的后端、OLMES / OpenCompass 的兼容标准、frontier lab 内测的 reference 工具。任何在它之上的二次开发——加 task、换 backend、debug 数字偏差——都绕不开三个目录：`lm_eval/api/`、`lm_eval/tasks/`、`lm_eval/models/`。Biderman 等 2024 paper 把它定位为对 prompt 微扰高敏感性、跨方法学不可比、复现性不足三类问题的对策[^biderman2024lessons]，但同时承认默认 setup 仍开放——读源码是用好它的前提，不是可选。

当前 stable v0.4.12（2026-05-11 release）。下文给出的文件路径以 `main` 分支为准；行号在 0.4.x 系列内基本稳定。

目录骨架

```
lm-evaluation-harness/
├── lm_eval/
│   ├── api/                  # 抽象层：Task / Model / Instance / Filter
│   │   ├── task.py           # ConfigurableTask 主入口
│   │   ├── model.py          # LM 抽象基类（loglikelihood / generate_until / ...）
│   │   ├── instance.py       # Instance 数据类
│   │   ├── filter.py         # 输出后处理 filter
│   │   ├── metrics.py        # acc / acc_norm / exact_match / pass@k
│   │   ├── registry.py       # @register_task 装饰器
│   │   └── samplers.py       # few-shot 采样
│   ├── tasks/                # YAML + Python 的 600+ task 定义
│   │   ├── mmlu/             # mmlu_*.yaml + default/_default_template_yaml
│   │   ├── hellaswag/        # hellaswag.yaml + utils.py
│   │   ├── gsm8k/            # gsm8k-cot.yaml + gsm8k.yaml
│   │   ├── arc/              # arc_challenge.yaml / arc_easy.yaml
│   │   ├── humaneval/        # humaneval.yaml + execution sandbox
│   │   └── leaderboard/      # HF v2 leaderboard task 集合
│   ├── models/               # backend
│   │   ├── huggingface.py    # HF Transformers reference impl
│   │   ├── vllm_causallms.py # vLLM
│   │   ├── openai_completions.py
│   │   ├── anthropic_llms.py
│   │   ├── sglang.py
│   │   └── nemo_lm.py
│   ├── evaluator.py          # 顶层调度
│   ├── __main__.py           # `lm_eval` CLI 入口
│   └── filters/              # regex / extraction filter
└── scripts/
```

90% 的 task 改动只动 `lm_eval/tasks/<task>/<task>.yaml`；10% 涉及自定义 metric 或 doc 处理时才需要 Python 文件。

Task YAML 范式

最小可跑 task：

```
# lm_eval/tasks/arc/arc_challenge.yaml
task: arc_challenge
dataset_path: allenai/ai2_arc
dataset_name: ARC-Challenge
output_type: multiple_choice
training_split: train
validation_split: validation
test_split: test
doc_to_text: "Question: {{question}}\nAnswer:"
doc_to_target: "{{choices.label.index(answerKey)}}"
doc_to_choice: "{{choices.text}}"
should_decontaminate: true
doc_to_decontamination_query: "Question: {{question}}\nAnswer:"
metric_list:
  - metric: acc
    aggregation: mean
    higher_is_better: true
  - metric: acc_norm
    aggregation: mean
    higher_is_better: true
metadata:
  version: 1.0
```

字段含义：

- `output_type`：四选一 `loglikelihood` / `loglikelihood_rolling` / `generate_until` / `multiple_choice`。决定 task instance 翻译成哪种 Request。
- `doc_to_text` / `doc_to_target`：Jinja2 模板，注入 dataset 列。注意 `{{ }}` 内的空格、末尾换行符直接影响 tokenization——MMLU 历史上一次错放 `\n` 导致主流模型分数下移 2 pp 的 bug 在 issue #1196 修过。
- `doc_to_choice`：仅 `multiple_choice`，给出候选列表，harness 自动为每条 doc 生成 N 个 loglikelihood request。
- `metric_list`：注册到 `lm_eval/api/metrics.py` 里的 metric。`acc_norm` 用 byte-length normalize log-likelihood，对 HellaSwag / ARC 这种候选长度差异大的任务有效。
- `should_decontaminate` / `doc_to_decontamination_query`：开启 n-gram contamination 检测（需要 `--decontamination_ngrams_path` 参数）。
- `metadata.version`：bump 后会在结果 JSON 里标 `versions` 字段，1.0 → 2.0 的数字不允许直接比。

6 / 64

`_default_template_yaml` 继承

MMLU 57 子集每个对应一个 YAML，但共用模板。
`lm_eval/tasks/mmlu/default/_default_template_yaml`：

```
dataset_path: hails/mmlu_no_train
test_split: test
fewshot_split: dev
output_type: multiple_choice
doc_to_text: "{{question.strip()}}\nA. {{choices[0]}}\nB. {{choices[1]}}\nC. {{c
doc_to_choice: ["A", "B", "C", "D"]
doc_to_target: answer
metric_list:
  - metric: acc
    aggregation: mean
    higher_is_better: true
metadata:
  version: 1.0
```

子集 `mmlu_abstract_algebra.yaml`：

```
include: _default_template_yaml
task: mmlu_abstract_algebra
task_alias: abstract_algebra
dataset_name: abstract_algebra
description: "The following are multiple choice questions (about abstract algebr
```

`include` 关键字层叠合并；`description` 在 `doc_to_text` 前拼接，是 5-shot prompt 的 system 段。这是 OLMES paper 反复批评的"开放选择"：harness 把 prompt template 暴露给 task author，自由选 description 措辞、`Answer:` 后是否带空格、选项是 `A.` 还是 `(A)` 还是 `A:`，每一处都可能在不同模型上有 1–3 pp 漂移[^gu2024olmes]。

`mmlu_*` 的 group 配置在 `lm_eval/tasks/mmlu/default/_mmlu.yaml`：

7 / 64

```
group: mmlu
group_alias: MMLU
task:
  - group: mmlu_stem
    task:
      - mmlu_abstract_algebra
      - mmlu_anatomy
      ...
  - group: mmlu_humanities
  - group: mmlu_social_sciences
  - group: mmlu_other
aggregate_metric_list:
  - metric: acc
    aggregation: mean
    weight_by_size: true
```

`weight_by_size: true` 是 MMLU 标准 aggregation——subset 按样本数加权平均，而非简单 arithmetic mean。早期 huggingface/leaderboards repo 一段时间用 simple mean，与 paper 不一致，2024 年 PR #2049 修正。

Request 路径：从 doc 到 LM 调用

`evaluator.simple_evaluate` 在 `lm_eval/evaluator.py` 里编排：

```
# 简化
task_dict = get_task_dict(task_names, task_manager)
for task_name, task in task_dict.items():
    task.build_all_requests(
        limit=limit, rank=rank, world_size=world_size,
        cache_requests=cache_requests, rewrite_requests_cache=rewrite_requests_c
        system_instruction=system_instruction, apply_chat_template=apply_chat_te
        fewshot_as_multiturn=fewshot_as_multiturn, chat_template=chat_template,
        tokenizer_name=tokenizer_name,
    )
    requests = task.instances
# group by request type, batch dispatch to LM
for reqtype in ["loglikelihood", "loglikelihood_rolling", "generate_until"]:
    cloned_reqs = [r for inst in all_instances for r in inst.requests if r.reque
    resps = getattr(lm, reqtype)([r.args for r in cloned_reqs])
    # write back
```

关键调用 `task.build_all_requests` 在 `lm_eval/api/task.py:1100+`：每个 doc 触发：

1. `fewshot_context()` 拼出 few-shot prompt（含 `description` + N 个 example + 当前 query）。

2. `construct_requests(doc, ctx)` 根据 `output_type` 生成 1 个或 N 个 `Instance(request_type, arguments, ...)`。

对 `multiple_choice` 的 ARC 题目，构造 4 个 `Instance(request_type="loglikelihood", arguments=(ctx, " A"))` 等候选——注意候选前置空格 `" A"` 而非 `"A"`，因为 tokenization 边界——`Answer:` 后接 `A` 通常是一个独立 token，接 `A` 则可能被合并到 `Answer:A`。harness 默认带空格，但部分 fork（如某些 Chinese benchmark 自家魔改）不带，是数字差异常见根因。

对 `generate_until` 的 GSM8K，构造 1 个 `Instance(request_type="generate_until", arguments=(ctx, {"until": ["\n\n", "Question:"], "do_sample": false, "max_gen_toks": 256}))`。`until` 列表是 stop sequence，第一个匹配即截断。

LM 抽象：四个方法

`lm_eval/api/model.py:LM` 定义抽象基类：

```
class LM(abc.ABC):
    @abc.abstractmethod
    def loglikelihood(self, requests: list[Instance]) -> list[tuple[float, bool]
        """returns list of (logprob, is_greedy) tuples."""
        ...

    @abc.abstractmethod
    def loglikelihood_rolling(self, requests: list[Instance]) -> list[float]:
        """rolling perplexity over long context (LAMBADA, wikitext)."""
        ...

    @abc.abstractmethod
    def generate_until(self, requests: list[Instance]) -> list[str]:
        """autoregressive generation till stop sequence."""
        ...

    def apply_chat_template(self, chat_history: list[dict]) -> str:
        ...
```

`loglikelihood(ctx, continuation)` 返回两元组 `(logprob, is_greedy)`：

- `logprob`：在 `ctx` 上下文里 `continuation` 的对数概率和，不做 length normalize（normalize 在 task 侧 `process_results` 做）。
- `is_greedy`：True 当且仅当 continuation 的每个 token 都是 argmax。这个字段在 perplexity-style 任务有意义（衡量"贪心解码会不会自然吐出这段 continuation"），但 MCQ 任务直接忽略。

`generate_until(ctx, kwargs)` 返回字符串，已截到 stop sequence 前。注意：harness 默认 stop sequence 是 task YAML 里的 `generation_kwargs.until`；如果模型

在 stop sequence 前就吐了 EOT，截断到 EOT；如果一路生成到 `max_gen_toks` 还没 hit，返回完整字符串（task 侧 filter 自己处理）。

HF backend 内部：`_loglikelihood_tokens`

`lm_eval/models/huggingface.py:_loglikelihood_tokens` 是 reference 实现，所有数字比对以它为准：

```
# lm_eval/models/huggingface.py 约 800-900 行
def _loglikelihood_tokens(self, requests, disable_tqdm=False, override_bs=None):
    res = []
    # padding 策略：按 (context+continuation) 总长降序排，减少 padding waste
    re_ord = utils.Reorderer(requests, _collate)
    chunks = re_ord.get_batched(n=adaptive_batch_size, batch_fn=...)
    for chunk in chunks:
        inps, cont_toks_list, inplens = [], [], []
        for _, context_enc, continuation_enc in chunk:
            inp = torch.tensor((context_enc + continuation_enc)[-(self.max_lengt
            inplens.append(inp.shape[0])
            inps.append(inp)
        batched_inps = utils.pad_and_concat(self.max_length, inps, padding_side=
        multi_logits = F.log_softmax(self._model_call(batched_inps), dim=-1)
        for (cache_key, _, _), logits, inp, inplen, cont_toks in zip(chunk, mult
            contlen = len(cont_toks)
            logits = logits[inplen-contlen:inplen].unsqueeze(0)
            greedy_tokens = logits.argmax(dim=-1)
            cont_toks_t = torch.tensor(cont_toks, dtype=torch.long).unsqueeze(0)
            max_equal = (greedy_tokens == cont_toks_t).all()
            logits = torch.gather(logits, 2, cont_toks_t.unsqueeze(-1)).squeeze(
            answer = (float(logits.sum()), bool(max_equal))
            res.append(answer)
    return re_ord.get_original(res)
```

注意四个细节：

1. **`[-(max_length+1):][:-1]`** 把 input truncate 到 `max_length`，从右截断——超过模型 context window 的 context 会丢前缀；GSM8K few-shot prompt 接近 4k token 时这步会悄悄丢 example，模型数字偏低且 log 里没有 warning。
2. **`F.log_softmax(...)` 前调用 `_model_call`** 拿 logits；对 dtype=bf16 模型，log_softmax 在 fp32 upcast 与否会有 1e-5 量级差异，与 vLLM backend 的 0.2 pp 漂移源头之一。
3. **`logits[inplen-contlen:inplen]`** 只取 continuation 那段 logit。这里 off-by-one 是早期 bug 高发区。
4. **`max_equal = (greedy_tokens == cont_toks_t).all()`** 是 `is_greedy` 来源。MCQ 任务忽略，但 `loglikelihood_rolling` 任务（LAMBADA）依赖它判断 "exact-match-on-greedy"。

10 / 64

HellaSwag 的 acc_norm：byte-length normalize 的工程实现

HellaSwag 是 `acc_norm` 的代表性 task。
`lm_eval/tasks/hellaswag/hellaswag.yaml`：

```
task: hellaswag
dataset_path: hellaswag
output_type: multiple_choice
training_split: train
validation_split: validation
process_docs: !function utils.process_docs
doc_to_text: "{{query}}"
doc_to_target: "{{label}}"
doc_to_choice: "choices"
metric_list:
  - metric: acc
    aggregation: mean
    higher_is_better: true
  - metric: acc_norm
    aggregation: mean
    higher_is_better: true
metadata:
  version: 1.0
```

`process_docs` 指向 `lm_eval/tasks/hellaswag/utils.py`，把原 dataset 的 `ctx_a / ctx_b / activity_label` 三段拼成最终 `query`，并把四个 endings 编入 `choices`。
`acc_norm` 的计算在 `lm_eval/api/task.py:process_results`：

```
# multiple_choice 的 process_results 简化
ll, is_greedy = zip(*results)  # 四个候选的 loglikelihood
ll_norm = [l / len(c) for l, c in zip(ll, doc["choices"])]  # 按 byte 长度 normal
pred = int(np.argmax(ll))
pred_norm = int(np.argmax(ll_norm))
return {
    "acc": float(pred == doc["label"]),
    "acc_norm": float(pred_norm == doc["label"]),
}
```

`/ len(c)` 这一步在 byte 单位上做 normalize（用 Python `len(str)` 而非 `len(tokenizer.encode(c))`——这是 EleutherAI 在 blog 里反复澄清的点[^eleuther-mcq-norm]）。HellaSwag 候选续写长度差异大（最短 5 词、最长 25 词），不 normalize 时模型几乎总选最短候选，acc 与 acc_norm 通常差 10–15 pp。

EleutherAI 自家 blog 实测：byte / char / unconditioned / acc_norm 四种 normalize 在同一模型同一 task 上数字差 5–15 pp[^eleuther-mcq-norm]。OLMES 把 cloze-formulation task

的 normalize 钉为 character-length；harness 默认是 byte-length——两数字不能直接比。

vLLM backend 的差异点

`lm_eval/models/vllm_causallms.py:VLLM.loglikelihood` 调 vLLM 的 `LLM.generate(..., prompt_logprobs=1)`，把 prompt 段的 token-level logprob 一次性拿回。优点是 paged-attention 大幅提速；代价：

- **Chunked prefill** 在长 context 时把 prompt 分块算 attention，与 HF 一次性 attention 在数值上有 1e-4 量级偏差。
- **Tensor parallel + reduction order** 不同 GPU 数下 all-reduce 顺序不一致，bf16 累加非交换；在 MMLU 14042 题上 score 抖动 ±0.1–0.3 pp。
- **`prompt_logprobs=1`** 只返回每位置 top-1 logprob，sum 时按 ctx+cont token id 索引——若 tokenizer 对 continuation 前导空格处理不同（HF Llama tokenizer 偶有 `" A"` 编成 `[259, 32]` vs `[284]`），数字不对。

Biderman 等明示：**正式 leaderboard 数字仍用 HF backend**，vLLM 仅作 sanity check[^biderman2024lessons]。一个常见 workflow：开发时 vLLM 跑全套 + 抓信号；release 前 HF 重跑顶层 task 复核。

generate_until 的隐患

GSM8K task YAML（`lm_eval/tasks/gsm8k/gsm8k-cot.yaml`）：

```
task: gsm8k_cot
dataset_path: gsm8k
dataset_name: main
output_type: generate_until
training_split: train
test_split: test
fewshot_split: train
doc_to_text: "Q: {{question}}\nA:"
doc_to_target: "{{answer.split('####')[-1].strip()}}"
generation_kwargs:
  until:
    - "Q:"
    - "</s>"
    - "<|im_end|>"
  do_sample: false
  temperature: 0.0
filter_list:
  - name: "strict-match"
    filter:
      - function: "regex"
        regex_pattern: "#### (\\-?[0-9\\.\\,]+)"
      - function: "take_first"
  - name: "flexible-extract"
    filter:
      - function: "regex"
        group_select: -1
        regex_pattern: "(-?[$0-9.,]{2,})|(-?[0-9]+)"
      - function: "take_first"
metric_list:
  - metric: exact_match
    aggregation: mean
    higher_is_better: true
    ignore_case: true
    ignore_punctuation: false
    regexes_to_ignore:
      - ","
      - "\\$"
      - "(?s).*#### "
      - "\\.$"
num_fewshot: 8
metadata:
  version: 3.0
```

两个 filter 给出 `exact_match,strict-match` 与 `exact_match,flexible-extract` 两个分数——前者要求模型输出严格 `#### 42` 格式，后者从最后一个数字 token 抽取。Llama-3.1-8B-Base 在 GSM8K 上常见 strict-match 0.5072 / flexible-extract 0.5375——~3 pp 的差距全是格式：base 模型在 CoT 末尾不写 `####` 。Paper 报数字时务必明示 filter（OLMo 3 / Llama 系列默认用 strict，但 OpenLM 系列倾向 flexible）。

13 / 64

`until` 列表的设计：第一个匹配的停止；写错顺序或忘记 `</s>` 会让 instruct 模型生成完毕后继续生成下一道题。常见 bug：跑 chat 模型时漏 `<|im_end|>` 或 `<|eot_id|>`，wall-clock 翻 3 倍且分数无变化（被 stop sequence 之后的内容拖累，但 filter 抽到首个数字侥幸正确）。

Few-shot 的种子陷阱

`lm_eval/api/samplers.py` 的 few-shot 采样：

```
class ContextSampler:
    def __init__(self, docs, task, fewshot_indices=None, rnd=None):
        self.docs = docs
        self.rnd = rnd or random.Random(self.fewshot_seed)
        ...

    def sample(self, n: int, seed: int) -> list[doc]:
        rnd = random.Random(seed)
        return rnd.sample(self.docs, n)
```

`fewshot_seed` 在 `lm_eval/__main__.py` 通过 CLI `--seed` 透传。默认 `--seed 0,1234,1234,1234`（fewshot seed = 1234）。但**不同 lm-eval 版本 default 不同**：v0.3.x 的 default 是 0，v0.4.x 改为 1234。同一模型同一 task 跨版本可能 0.5–1.5 pp 漂移就因为这个。OLMES 的 canonical setup 强制固定 example 列表（不允许随机），就是为绕开此问题[^gu2024olmes]。

把 seed 固定 + 把 `--log_samples` 打开，重跑差异在 0.001 pp 量级；任何更大的差异都是 backend / harness 版本 bug。

Chat template 与 base 模型的歧义

`--apply_chat_template` 在 0.4.4 之后开放。
`lm_eval/api/task.py:fewshot_context` 大概逻辑：

```
if apply_chat_template:
    if fewshot_as_multiturn:
        # 把 N 个 few-shot example 翻译成 N 轮 user/assistant turn
        chat_history = []
        for ex in fewshot_examples:
            chat_history.append({"role": "user", "content": self.doc_to_text(ex)
            chat_history.append({"role": "assistant", "content": self.doc_to_tar
        chat_history.append({"role": "user", "content": self.doc_to_text(doc)})
        ctx = lm.apply_chat_template(chat_history)
    else:
        # 把整段 few-shot prompt 当一个 user turn
        prompt = self._build_legacy_fewshot_prompt(doc, fewshot_examples)
        ctx = lm.apply_chat_template([{"role": "user", "content": prompt}])
else:
    ctx = self._build_legacy_fewshot_prompt(doc, fewshot_examples)
```

`fewshot_as_multiturn` 决定 few-shot example 是单 turn 拼成长 prompt，还是 N 轮多 turn——HF Open LLM Leaderboard v2 chat 模型默认开 `fewshot_as_multiturn` [^hf-leaderboard]，与 base 模型的"长 prompt 一气呵成"setup 不同。两数字在 IFEval / BBH 上可能差 3–5 pp。

针对 base ckpt，**永远不应 `--apply_chat_template`**——base 模型从未见过 `<|im_start|>system\n` 这类 token，logprob 在这些 token 上是 noise，会把 MCQ task 数字打下来 5–20 pp。Biderman 等的 paper 第 4 节明示此类 chat-template fallback 是新人最常踩的坑[^biderman2024lessons]。

多 task 调度与 Reorderer

`lm_eval/utils.py:Reorderer` 是 throughput 关键。原理：把所有 `(context, continuation)` 按 `len(context) + len(continuation)` 降序排，padding 浪费最小化；最长的一组先跑，避免末尾"长 sample 落单"。

```
class Reorderer:
    def __init__(self, arr, fn):
        # arr: list of items; fn(item) → sort key
        self.size = len(arr)
        self.arr = sorted([(i, x) for i, x in enumerate(arr)], key=lambda kv: -f

    def get_reordered(self):
        return [x for _, x in self.arr]

    def get_original(self, newarr):
        # 重排回原始顺序
        res = [None] * self.size
        for (i, _), v in zip(self.arr, newarr):
            res[i] = v
        return res
```

在 8 GPU vLLM 上跑 MMLU 14,042 题（每题 4 候选，56,168 个 loglikelihood request），Reorderer 把 batched throughput 从 ~280 req/s 提到 ~520 req/s（实测，A100 80G × 8 + Llama-3.1-8B + bf16）。`--batch_size auto` 与 Reorderer 配合，从 batch_size=1 开始 binary-search 到 OOM 前的最大值，自动锁档。

DataParallel / 多 process scale-out

`lm-eval` 没有内置 distributed runner，靠 backend 自带 parallelism：

- **HF backend**：`--model_args parallelize=True` 触发 `transformers.AutoModelForCausalLM.from_pretrained(..., device_map="auto")` 的 pipeline 切分。简单但慢。
- **vLLM backend**：`--model_args tensor_parallel_size=8` 触发 vLLM 的 TP；数据并行靠多 process。
- **跨节点**：lm-eval 自己跑不起；典型做法是按 task 拆 process（task A 在 node 1、task B 在 node 2），最后合并 `results_*.json`。`scripts/merge_results.py` 是常用工具。

`accelerate launch --num_processes 8 -m lm_eval.__main__ --model hf ...` 也能跑，但 HF backend 的 multi-GPU loglikelihood 路径在 0.4.x 仍偶有 deadlock（GitHub issue #1832 跟踪），不如 vLLM 稳。

Caching：重跑成本下降的关键

`--cache_requests` 把 `(task, doc_id, request)` 三元组的结果缓存到 `~/.cache/lm-eval/`。重跑同模型同 task 时，loglikelihood 与 generate_until 都走 cache hit；ckpt 不变情况下重跑 GSM8K 从 ~25 min 降到 ~5 s（纯磁盘 IO + metric 计算）。

注意：cache key 不含 `model_args`——同 ckpt 用 vLLM 跑过 cache，再用 HF 跑会读 cache 而非重算。`--rewrite_requests_cache` 强制 invalidate。开发期定期清 cache 是好习惯。

Filter / answer extraction 的 anti-pattern

`lm_eval/filters/` 下有 `regex_filter.py`、`extraction.py`、`selection.py`、`take_first.py`。常见 anti-pattern：

- **贪心 regex** `"\\d+"` 在 CoT 输出 "First, 3 + 5 = 8. Then 8 * 2 = 16. So the answer is 16" 上匹配到 `3`（首个），不是 `16`。修法：用 `group_select: -1` 取最后一个，或写更窄的 regex `"(?:answer is|####)\\s*(-?\\d+)"`。
- **大小写敏感**：`ignore_case: true` 默认关；EM 比 "Paris" vs "paris" 会判 fail。
- **Unicode normalize**：模型输出全角逗号 `，`，ground truth 半角 `,`；EM fail。`regexes_to_ignore` 列表里加 `","` 不够，还得在 Python 侧 NFC normalize（部分 task 在 `utils.py` 里做）。

16 / 64

HumanEval 的 sandbox 在 `lm_eval/tasks/humaneval/utils.py`：

```
def pass_at_k(n, c, k):
    if n - c < k:
        return 1.0
    return 1.0 - np.prod(1.0 - k / np.arange(n - c + 1, n + 1))

def process_results(doc, results):
    # results: list of model outputs
    # 注意 humaneval 默认 execution=False，需要 ALLOW_CODE_EXECUTION=1
    if os.environ.get("ALLOW_CODE_EXECUTION", "0") != "1":
        raise PermissionError("HumanEval execution requires ALLOW_CODE_EXECUTION=
    ...
```

默认禁用代码执行——任何首次跑 HumanEval 的人都会撞墙。export 这个环境变量后才会真正进 sandbox。容器化 sandbox（`subprocess` + `signal.SIGKILL` timeout）是 lm-eval 自带的最弱实现，大规模 leaderboard 通常自己接 BigCode `bigcode-evaluation-harness` 或 EvalPlus。

常见 bug 与诊断

工程实操中按下列优先级查：

1. **数字与 paper 差 >2 pp**：先看 `versions` 字段是否同；再看 `num_fewshot` 与 `description` 是否同；再 `--apply_chat_template` 对错；再 backend 是否 vLLM/HF。
2. **MCQ 全选 A**：tokenizer pad token 没设，batched inference 把 padding 当真 token 算 logprob，max 选第一个候选。`tokenizer.pad_token_id = tokenizer.eos_token_id` 解决。
3. **GSM8K wall-clock 异常慢**：stop sequence 漏 chat template 的 `<|im_end|>` / `<|eot_id|>`，模型生成到 `max_gen_toks`。在 `samples_*.jsonl` 里抽几条 `resps` 字段看长度。
4. **HumanEval pass@1 = 0**：`ALLOW_CODE_EXECUTION=1` 没开，所有 sample 走 `PermissionError` 路径。
5. **MMLU 子任务数字奇怪**：subset 名拼错（`mmlu_us_foreign_policy` vs `mmlu_high_school_us_history`），harness 静默跳过——结果 JSON 里少 subset，主分数被错误平均。

在 0.4.x 之后的演化

lm-eval-harness 在 issue #2002 跟踪吸收 OLMES 标准的进展——具体讨论：normalization 默认是否切到 `acc_norm`、few-shot example pool 是否固定、`apply_chat_template` 是否对 base 模型 explicit 报错。截至 v0.4.12，部分 OLMES 推荐已 backport（如 cloze

formulation 的 `acc_norm` 默认），其余仍在讨论[^biderman2024lessons]。OLMES 自身也以 `allenai/olmes` 包外挂在 harness 之上——这一点会在第 3 章框架对比章详谈。

读到这一步，你已经能：写新 task YAML 时不踩边界 bug、能在 `samples_*.jsonl` 里 audit 任意一条样本的全链路、能解释为什么自家数字与 paper 差 X pp。这是工程实践的最小工作底层。

引用

[^biderman2024lessons]: Biderman, S. et al. (2024). *Lessons from the Trenches on Reproducible Evaluation of Language Models*. arXiv:2405.14782.

[^gu2024olmes]: Gu, Y., Tafjord, O., Kuehl, B., Haddad, D., Dodge, J., Hajishirzi, H. (2024). *OLMES: A Standard for Language Model Evaluations*. arXiv:2406.08446 (Findings of NAACL 2025).

[^eleuther-mcq-norm]: EleutherAI Blog. *Multiple Choice Normalization in LM Evaluation*. https://blog.eleuther.ai/multiple-choice-normalization/.

[^lm-eval-repo]: EleutherAI. *lm-evaluation-harness*. https://github.com/EleutherAI/lm-evaluation-harness.

[^lm-eval-issue-2002]: EleutherAI. *Implementing lessons from OLMES (issue #2002)*. https://github.com/EleutherAI/lm-evaluation-harness/issues/2002.

[^hf-leaderboard]: HuggingFace. (2024-06-26). *Open-LLM performances are plateauing, let's make the leaderboard steep again (Open LLM Leaderboard v2 blog)*. https://huggingface.co/spaces/open-llm-leaderboard/blog. 2025-03-13 retirement: https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard/discussions/1135.

为什么要四个框架

读过第 2 章会有一个直觉问题：lm-eval-harness 已经"什么 task 都收"，为何还需要 OLMES / lighteval / HELM v2 / DCLM？答案在四件事上分别裂开：

维度	lm-eval-harness	OLMES	lighteval	HELM v2	DCLM
主线问题	工具齐全	标准化	易扩展 + 多 backend	评分卡	数据评估
Execution model	单 process / vLLM	单 process（基于 harness）	accelerate + nanotron / vLLM	distributed runner	OpenLM training-time eval
Prompt 模板	开放	强制 canonical setup	inherit OLMES + 自定	scenario-specific adapter	固定 53-task
Metric 抽象	task 自定	task 固定	task 兼容 harness	7 维 metric matrix	53-task macro avg
Scale-out	DataParallel	同 harness	accelerate FSDP / DP	manual sharding	OpenLM cluster
适用场景	单 ckpt eval	reproducible release	HF 训练栈集成	holistic 报告	data recipe ablation

下面拆开看。

OLMES：在 harness 之上加锁

OLMES（Open Language Model Evaluation System，Gu et al. 2024 / NAACL 2025[^gu2024olmes]）的工程定位是"harness compatible + 强制 canonical setup"。它的 `allenai/olmes` 包不重写 evaluation engine——内部直接依赖 lm-eval-harness，但替换 task YAML 与配置：每个 task 的 prompt template / few-shot example / normalization choice / stop sequence 都被钉死。

跑一次：

```
pip install ai2-olmes
olmes \
  --task arc_challenge::olmes mmlu::olmes gsm8k::olmes hellaswag::olmes \
  --model meta-llama/Llama-3.1-8B \
  --model-type vllm \
  --output-dir ./olmo-eval-out \
  --num-shots 5 \
  --gpus 8
```

`task::olmes` 的 `::olmes` 后缀是 OLMES task suite 命名空间；它会到 `olmes/tasks/<task>_olmes.yaml` 找配置，覆盖 harness 同名 task 的 prompt template。AI2 在 OLMo 3 release 时同时打包 **OlmoBaseEval**——43 个 benchmark，按 "Base Easy"（小规模 pretraining run proxy）/ "Base Main"（full-scale run）分层，与 harness 的 task 完全独立维护[^olmo3-blog]。

OLMES 的核心工程贡献是"instance-level logging"：每条样本写一行到 `predictions.jsonl`：

```
{"task": "arc_challenge::olmes", "instance_id": "Mercury_7175875",
 "ctx": "Question: ...\nA. ...\nB. ...\nAnswer:",
 "choices": ["A","B","C","D"],
 "logprobs": [-3.12, -5.61, -2.04, -4.89],
 "pred": "C", "gold": "C", "correct": true,
 "metadata": {"few_shot_seed_fixed": true, "norm": "char_length", "version": "ol
```

可直接灌 Google Sheets / W&B / S3。这是 lm-eval-harness `--log_samples` 的"产品化"版——不只是 jsonl 落盘，还做了 audit-friendly 的 schema。

OLMES 的代价：与 AI2 的 OLMo training workflow 绑定越来越深；社区中立性下降——OlmoBaseEval 43 个 benchmark 是 AI2 偏好的能力组合，他人用就要接受这套口味[^olmo3-blog]。`lm-evaluation-harness` 在 issue #2002 跟踪逐步吸收 OLMES 建议，但完整 backport 尚未完成；目前 dual-report（OLMES + harness default）是 OLMo / Llama 系列复现的最佳实践。

19 / 64

lighteval：HF 训练栈的原生 eval

lighteval（`huggingface/lighteval`）是 HuggingFace 团队 2024 年起维护的轻量评测框架，明确定位"HF 训练栈（accelerate / nanotron / TRL）的原生 eval"。设计上的差异点：

```
lighteval accelerate \
  --model_args "pretrained=meta-llama/Llama-3.1-8B,dtype=bfloat16" \
  --tasks "leaderboard|mmlu|0|0,leaderboard|hellaswag|0|0,leaderboard|gsm8k|5|0"
  --num_fewshot_seeds 1 \
  --output_dir ./lighteval-out \
  --override_batch_size 32
```

注意 task 语法 `<suite>|<task>|<num_fewshot>|<truncate>`——四元组直接嵌在 CLI，比 harness 的 `--num_fewshot` 全局参数更细粒度。lighteval 的 backend 有三：

- **`accelerate`**: HF accelerate distributed runner，原生支持 FSDP / DeepSpeed ZeRO。
- **`nanotron`**: nanotron 训练框架的 in-training eval（边训边评，无需 ckpt 落盘）。
- **`vllm`** / **`tgi`**: vLLM 或 TGI 推理 backend。
- **`endpoint`**: 任意 OpenAI-compatible HTTP endpoint。

lighteval 的 task 系统兼容 lm-eval-harness（`extended/lighteval_tasks.py` 里 import harness task 注册器），并支持 OLMES task 包做 drop-in 替换。它的"杀手特性"是 **in-training eval**：训练 loop 跑到指定 step 时直接调 lighteval 的 evaluator，无需把 ckpt 写 HF safetensors 再 load——这对 trillion-token pretrain 是关键省时点。

工程权衡：lighteval 的 task 库不如 harness 全（约 200 vs 600+ task），且 in-training eval 仍是 generative，跑一遍 ~30 min（与第 1 章 in-training-probing 的 ~3 min 是数量级差距[^liu2026probing]）。

HELM v2：scenarios + adapter 解耦

HELM v2（HELM Capabilities，CRFM 2025-03 release）走完全不同路线[^crfm2025helm-capabilities]。核心抽象是 `scenarios`（"什么任务"）× `adapters`（"如何把任务翻译成 LLM 调用"）× `metrics`（"评估什么 dimension"）三轴解耦。一个 run spec：

```
helm-run \
  --conf-paths run_specs.conf \
  --suite v2 \
  --max-eval-instances 200 \
  --num-train-trials 1 \
  --model anthropic/claude-3-5-sonnet-20241022 anthropic/claude-3-5-haiku-202410
  --output-path benchmark_output
helm-summarize --suite v2
helm-server --suite v2  # 启动本地 leaderboard 网页
```

`run_specs.conf` 长这样：

```
entries: [
  {description: "mmlu_pro:subject=all,model=text", priority: 1}
  {description: "gpqa:subset=main,model=text", priority: 1}
  {description: "ifeval:model=text", priority: 1}
  {description: "wildbench:model=text", priority: 1}
  {description: "omni_math:model=text", priority: 1}
]
```

每个 entry 是 scenario 的可参数化引用；HELM runner 把它 expand 成 (scenario instance, adapter, metric) 三元组，分批跑。Adapter 决定 prompt format / 是否 CoT / few-shot 数；metric 决定 accuracy / robustness / calibration / efficiency / fairness / bias / toxicity 七维要哪几个。

HELM v2 把 v1 的 mean-win-rate aggregation 换成 mean-score——后者与 model 集无关，新模型加入不会扰动旧模型 rank[^crfm2025helm-capabilities]。但 v2 同时把 scenario 收缩到 5 个（MMLU-Pro / GPQA / IFEval / WildBench / Omni-MATH），coverage 大幅缩小，与原"holistic"愿景背离——这是 HELM 自身 critique 的核心点。

HELM v2 的 multi-LLM-judge（GPT-4o + Llama-3.1-405B + Claude-3.5-Sonnet）实现在 `helm/benchmark/annotation/`：

```
# helm/benchmark/annotation/wildbench_annotator.py 简化
def annotate(self, request_state):
    judge_prompts = [self._build_judge_prompt(request_state, judge) for judge in
    judge_responses = [self._call_judge(judge, p) for judge, p in zip(self.judge
    scores = [self._parse_score(r) for r in judge_responses]
    return {"score": float(np.mean(scores)), "raw_judges": judge_responses}
```

3 judge 取均值在 reproducibility 上引入新噪声：每次 judge call 不严格一致，每 task 数字的 within-run stderr ≥ 0.5 pp。WildBench / Omni-MATH 这类 judge-based scenario 因此**永远做不到 lm-eval-harness loglikelihood task 那种"两次跑数字完全一致"**——HELM 自己也承认这点。

2026-06-01 HELM 进入 maintenance mode[^helm-github]——CRFM 不再为它加新功能。HELM Classic（v1，42 scenarios × 7 metrics）仍是想"一份报告覆盖 7 维"项目的唯一选择，但社区 momentum 已转向 lm-eval-harness / OLMES / lighteval。

DCLM：把 eval 绑数据评估

DCLM（DataComp-LM, Li, Fang et al. 2024 NeurIPS[^li2024dclm]）的工程定位与上面三者正交：它不是"通用 eval framework"，而是"把 eval 当作 data recipe 验证的固定 dependent variable"。范式：

```
Fix:  OpenLM transformer + 训练配方 + 53-task evaluation
Vary: 数据 recipe（filter / dedupe / mix）
```

跑一次 DCLM submission：

```
# 1. 准备 data manifest
python -m dclm.dataset.preprocess \
  --input-dir s3://my-corpus/ \
  --output-dir s3://my-corpus-cleaned/ \
  --apply-filter fasttext_high_quality

# 2. 训练（OpenLM）
python -m training.training \
  --train-data s3://my-corpus-cleaned/manifest.jsonl \
  --scale 7b_2x \
  --train-num-samples 138_000_000_000

# 3. 53-task eval
python -m eval.eval_openlm_ckpt \
  --eval-yaml ./eval/heavy.yaml \
  --model-dir ./checkpoints/openlm-7b-2x \
  --output-file ./results.jsonl
```

`scale: 7b_2x` 是 DCLM 三档 scale 的中等档（412M / 1B / 7B；后缀 `_1x` / `_2x` 表示 Chinchilla / 2× Chinchilla token budget）。每档绑死 token 数：7B-2x 是 140B token。这"固定模型 + 固定 token + 固定 eval"的设计让 data recipe 成为唯一独立变量[^li2024dclm]。

53-task eval YAML（`dclm/eval/heavy.yaml`）：

```
tasks:
  - mmlu_zero_shot
  - mmlu_fewshot
  - arc_easy
  - arc_challenge
  - hellaswag
  - hellaswag_zeroshot
  - winograd
  - winogrande
  - openbookqa
  - piqa
  - boolq
  - copa
  - siqa
  - commonsense_qa
  - jeopardy
  - bigbench_qa_wikidata
  - bigbench_dyck_languages
  ...
core_subset:
  - mmlu_zero_shot
  - arc_challenge
  - hellaswag_zeroshot
  - jeopardy
  - ...
```

"Core" 22 task 子集是 high-signal 子集，用 centered macro-average 作 top-level metric。DCLM-Baseline（fastText quality classifier 过滤后的 corpus）在 7B-2x 上 MMLU 5-shot 64%，把同期 Falcon-7B 的 26.2% 直接拉开 38 pp[^li2024dclm]——这是 data engineering 杠杆的 strong evidence。

DCLM 的工程坑：

- **OpenLM 与 HF 不通**：训完的 ckpt 是 OpenLM 格式，要转 HF 才能进 lm-eval-harness。`dclm/scripts/convert_to_hf.py` 做转换，但 RoPE / SwiGLU 实现的细微差异偶有 0.5 pp 抖动。
- **53-task BHI 视角已部分 saturate**：BHI paper（2026-02）指出 DCLM 53-task 中部分 capability discrimination 接近 0[^zhu2026bhi]——HellaSwag / PIQA 之类对 7B 模型分辨度不够；data recipe ablation 的信号被稀释。
- **未覆盖 mid-train / SFT**：DCLM 纯 pretrain centric；现代 pipeline 含 mid-training（continued pretrain on 高质量子集）和 SFT，DCLM 无法 isolate 这些阶段。daVinci-LLM (2026-03[^qin2026davinci]) 明示自己是 "next generation"，扩到 L0-L9 data taxonomy 与 mid-training。

选哪个：一张决策表

按"项目阶段 + 主要诉求"选：

| 场景 | 首选 | 二选 | 理由 | |---|---|---|---| | Pretrain run 高频中间评测 | lighteval (nanotron in-training) + in-training-probing | OLMES Base Easy | wall-clock 与训练栈集成 | | Pretrain run milestone 评测 | OLMES Base Main | lm-eval-harness + leaderboard task | canonical setup 可发表 | | Pretrain release / paper | OLMES + lm-eval-harness dual report | + HELM Classic 7-metric matrix | 用 OLMES 的标准化 + 暴露 setup sensitivity | | Data recipe ablation | DCLM 53-task | daVinci-LLM L0-L9 框架 | 范式锁住 model + eval，data 为独立变量 | | Holistic 评分卡（7 维） | HELM Classic | – | 仅此一家做 calibration/robustness/fairness/efficiency 一体化 | | Code 评估 | bigcode-evaluation-harness | EvalPlus | lm-eval-harness HumanEval 是最弱实现 | | Frontier reasoning 评估 | 自家 in-house | lm-eval-harness leaderboard 任务集 | 标准框架无 thinking-token 原生支持 | | HF training 栈集成 | lighteval accelerate | lm-eval-harness | accelerate distributed + nanotron 桥接 |

实操建议：**OLMES + lm-eval-harness 双跑** 是当前 release-quality eval 的最稳路径。两个数字对照能暴露 setup sensitivity（OLMES canonical vs harness default），任一框架内部 bug 都能被另一个抓到。HF Open LLM Leaderboard 已 archive[^hf-leaderboard-retirement]——不再有"上 leaderboard 就一切搞定"的捷径，每个项目都要自建 dual-report 流程。

后续章节

第 4 章起按 benchmark 类别（经典 / reasoning / live / 长上下文+多模态+agent）给逐 benchmark 的命令、prompt、log 片段。所有命令默认 OLMES 或 lm-eval-harness 任选其一，标注差异处。第 5 章专门处理 reasoning era 的 thinking-token 工程问题——上述四个框架都还没原生支持。

引用

[^gu2024olmes]: Gu, Y., Tafjord, O., Kuehl, B., Haddad, D., Dodge, J., Hajishirzi, H. (2024). *OLMES: A Standard for Language Model Evaluations*. arXiv:2406.08446 (Findings of NAACL 2025).

[^olmo3-blog]: Allen Institute for AI. (2025-11-20). *Olmo 3: Charting a path through the model flow to lead open-source AI*. https://allenai.org/blog/olmo3.

[^crfm2025helm-capabilities]: CRFM. (2025-03-20). *HELM Capabilities*. https://crfm.stanford.edu/2025/03/20/helm-capabilities.html.

[^helm-github]: Stanford CRFM. *HELM enters maintenance mode 2026-06-01*. https://github.com/stanford-crfm/helm.

[^li2024dclm]: Li, J., Fang, A., Smyrnis, G. et al. (2024). *DataComp-LM: In search of the next generation of training sets for language models*. NeurIPS 2024 Datasets and Benchmarks. arXiv:2406.11794.

[^zhu2026bhi]: Zhu, L., Hua, H., Miao, L., Zhao, B. (2026-02). *Benchmark Health Index: A Systematic Framework for Benchmarking the Benchmarks of LLMs*. arXiv:2602.11674.

[^qin2026davinci]: Qin, Y., Liu, Y., Mi, T., Xie, M., Huang, Z. et al. (2026-03). *daVinci-LLM: Towards the Science of Pretraining*. arXiv:2603.27164.

[^liu2026probing]: Liu, Z., Lun, T., Wen, Z. et al. (2026). *Fast and Accurate Probing of In-Training LLMs' Downstream Performances*. arXiv:2604.01025.

[^hf-leaderboard-retirement]: clefourrier @ HuggingFace. (2025-03-13). *It's been a wild ride, folks :) (end of the Open LLM Leaderboard)*. https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard/discussions/1135.

本章是经典 11 个 benchmark 的"操作手册"。每个 benchmark 给一段最小可跑命令、一段真实 log 片段、几个 paper 数字对账步骤，以及最容易踩的两到三个坑。前置假设：读者已按第 2 章把 `lm-evaluation-harness >= v0.4.12` 装好，且能用 `--model vllm` backend 跑通 Llama-3.1-8B 的 MMLU。

工作目录假定为 `/scratch/eval/`，`results/` 子目录留给输出。所有命令都给 base ckpt 的 0-shot/few-shot 协议——instruct 模型评测在第 5 章再讨论。

MMLU：五-shot log-likelihood 的"标准动作"

```
lm_eval \
  --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B,tensor_parallel_size=8,dtype=b
  --tasks mmlu \
  --num_fewshot 5 \
  --batch_size auto \
  --output_path results/llama31-8b/mmlu \
  --log_samples \
  --seed 1234
```

跑完后 `results_*.json` 顶层会出现 57 个 subtask + 4 个 group + 1 个 `mmlu` 总分：

```
"mmlu": {
  "alias": "mmlu",
  "acc,none": 0.6810,
  "acc_stderr,none": 0.0037
},
"mmlu_abstract_algebra": {
  "alias": " - abstract_algebra",
  "acc,none": 0.32,
  "acc_stderr,none": 0.0469
}
```

harness 文件路径锚点：task YAML 在 `lm_eval/tasks/mmlu/default/_mmlu.yaml`，57 个 subtask 各有 `_<subject>.yaml`，共享 `_default_template_yaml`。group 配置在 `default/_mmlu.yaml`。

坑 1：`acc` 是 raw、`acc_norm` 是 byte-length normalize 的——MMLU 没有 `acc_norm`

新人最容易混淆 MMLU 与 HellaSwag/ARC 的输出字段。MMLU 的官方协议是把 `A` / `B` / `C` / `D` 四个**字母**当 continuation 算 logprob，长度都是 1 token（多数 BPE tokenizer 下），不需要长度归一化，于是 harness 不报 `acc_norm`。HellaSwag 候选续写长度差几十 token，必须 byte-length 归一化才能比，于是默认上 `acc_norm`。两个字段拼错就拼出一个永远不收敛的 leaderboard——别问怎么知道的。

坑 2：`normalize_logprobs` 与"答案是文本"两种协议数字差 2–5 pp

harness 默认对 MMLU 用 4 个**字母** continuation；OLMES MMLU 用的是 **answer text**（"A. The polynomial is..."）作为 continuation，再 byte-length 归一化。Gu et al. 2024 OLMES paper[^gu2024olmes] 测出同一个 7B model 这两套协议差 3.1 pp，落在 60.4% 与 63.5% 之间。报数前先看协议——尤其要和 paper 对账时，必须明确对面跑的是哪套。

EleutherAI 自家 blog 的 MCQ normalization 帖（`blog.eleuther.ai/multiple-choice-normalization/` [^eleuther-mcq-norm]）实测 byte / char / unconditioned / acc-norm 四种归一化在 MMLU 上数字差最大 14 pp，看 paper 数字前 grep 一下作者用了哪种。

坑 3：5-shot demonstrations 的 dev-set 顺序

MMLU 的 5-shot 取自每个 subject 的 dev split（每 subject 5 题，正好作 5-shot），harness 的 `fewshot_split: dev` 与 `num_fewshot: 5` 保证 deterministic。但若改成 `--num_fewshot 25`，dev 不够会从 train split 抽——而原始 Hendrycks repo 没有 train split，harness 会 fallback 到 validation，subject 间顺序差异导致 3–5 pp 浮动。复现 paper 数字时永远用 `--num_fewshot 5`。

与 paper 数字对账 checklist

- 协议：letter-continuation（harness 默认）vs answer-text-continuation（OLMES）— 差 2–5 pp
- 归一化：`acc` vs `acc_norm` vs OLMES `acc_per_char` — 差 1–3 pp
- few-shot：5-shot dev / 0-shot — 差 4–8 pp（小模型大）
- chat template：是否 `--apply_chat_template`（base 模型必须 `false`）— 差 5–20 pp
- MMLU-Redux：用了官方的 14042 题还是 5700 题清洗版？看 Gema et al. 2024[^gema2024mmluredux]
- subdomain 加权：lm-eval-harness 默认对 57 subject 取算术平均，部分 paper 用题数加权——总分差 0.3–0.8 pp。这种差异不在乎，但若想精确复现 paper 数字需对齐。

Llama-3.1-8B base 我们这套配置实测 `mmlu acc=0.6810`，与 Meta model card 报的 0.68 一致；若你跑出 0.65 以下，先看 chat template 开关。

看 log：判断 inference 没走偏

`samples_mmlu_abstract_algebra_*.jsonl` 每条 record 长这样（省略部分字段）：

```
{
  "doc_id": 0,
  "doc": {"question": "Find the degree for the given field extension Q(sqrt(2),
  "target": " B",
  "arguments": {
    "gen_args_0": {"arg_0": "Question: ... Answer: A\n\nQuestion: ... Answer:",
    "gen_args_1": {"arg_0": "Question: ... Answer: A\n\nQuestion: ... Answer:",
    "gen_args_2": {"arg_0": "Question: ... Answer: A\n\nQuestion: ... Answer:",
    "gen_args_3": {"arg_0": "Question: ... Answer: A\n\nQuestion: ... Answer:",
  },
  "resps": [[["-1.234", "False"]], [["-0.876", "False"]], [["-2.105", "False"]],
  "filtered_resps": [-1.234, -0.876, -2.105, -1.890],
  "acc": 1.0
}
```

`resps` 数组里四个 log-probability 对应 A/B/C/D，argmax 是 B（-0.876 最大），与 `target=" B"` 匹配，`acc=1.0`。看这条 log 验证：(1) 4 个 choice 都跑了；(2) logprob 都是负数（base log），不是被 underflow 成 -inf；(3) `target` 前面带空格——MMLU 协议是 `\nAnswer:` 后跟一个 token；空格少了/多了，整套数字 shift 1-3 pp。这种细节看不到、查 30 分钟也找不到 bug。

HellaSwag：常识续写的 `acc_norm` 之坑

```
lm_eval --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B,tensor_parallel_size=8 \
  --tasks hellaswag --num_fewshot 0 \
  --output_path results/llama31-8b/hellaswag --log_samples
```

输出：

```
"hellaswag": {
  "acc,none": 0.6112,
  "acc_stderr,none": 0.0049,
  "acc_norm,none": 0.7920,
  "acc_norm_stderr,none": 0.0040
}
```

主报数字是 `acc_norm`（byte-length 归一化），原因前面说了：四个候选续写长度差异很大，不归一化会偏向短答案。Task YAML：`lm_eval/tasks/hellaswag/hellaswag.yaml`，`metric_list` 同时定义 `acc` 和 `acc_norm` 两条 metric。

27 / 64

坑：social-context label 残留

HellaSwag train/val 集每题前面有一个 `activity_label`（如 `"Personal Care and Style"`），harness 在 `doc_to_text` 里默认拼到 prompt 里：

```
def doc_to_text(doc):
    ctx = doc["activity_label"] + ": " + doc["ctx_a"] + " " + doc["ctx_b"].capit
    return ctx
```

`lm_eval/tasks/hellaswag/utils.py:preprocess` 还会做 `re.sub` 把 `[header]` / `[step]` 之类的 wikiHow tag 清掉。如果你看 paper 数字与 harness 数字差 1–2 pp，大概率是 paper 不带 `activity_label` 或不去 wikiHow tag——HellaSwag 原 paper 把这些视为 input 的一部分，社区分歧延续至今。

第二个常见 surprise：HellaSwag validation 已被广泛公开 ≥6 年，且 36% 的题目本身含错（Surge AI 2024 audit[^surge2024hellabad]），更严重的是 Chizhov et al. 2025 的 GoldenSwag 工作[^chizhov2025goldenswag] 显示 ≥65% 的模型预测在删除 question 或用 Lorem ipsum 替换后**不变**——也就是说大半部分分数靠"看答案选项"就能拿。把 HellaSwag 分数作为 saturation 后的 sanity check 还行，作为 ranking 信号 noise 已经压住 signal。

WinoGrande：paired log-likelihood 的 binary MCQ

```
lm_eval --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B,tensor_parallel_size=8 \
  --tasks winogrande --num_fewshot 5 \
  --output_path results/llama31-8b/winogrande --log_samples
```

输出：

```
"winogrande": {
  "acc,none": 0.7372,
  "acc_stderr,none": 0.0124
}
```

WinoGrande 是 binary（两个候选词填 `_`），random baseline = 50%；不需要 length-normalize 因为两句只差一个词。Task YAML：`lm_eval/tasks/winogrande/default.yaml`，`doc_to_choice` 直接返回 `[doc["option1"], doc["option2"]]`。

坑：5-shot vs 0-shot 在 7B 模型上差 8 pp

WinoGrande 短句 + binary，0-shot 时 7B 模型很容易 fall back 到 word-frequency baseline（约 53%）；加 5-shot demonstration 大跳到 70%+。这是为什么 Open LLM Leaderboard v1 用 5-shot 而早期 GPT-3 paper Table 3.6 用 0-shot——读数字时一定要确认 shot 数。

坑：test set label 不公开，社区报 validation

WinoGrande 有 train (40k) / dev (1267) / test (1767)，test label 维护在 AI2 leaderboard server（URL 已迁移到 GitHub Pages，不一定 active）。所有 LLM 论文实际报 dev set 数字。但若你看 hf-datasets 的 dataset card，可能误以为 test 是公开的——`split: test` 拉下来 label 全是 `-1`。

GSM8K：8-shot CoT 与 strict-match 的双指标

```
lm_eval --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B,tensor_parallel_size=8 \
  --tasks gsm8k --num_fewshot 8 \
  --batch_size auto \
  --output_path results/llama31-8b/gsm8k --log_samples
```

输出两条 EM 字段：

```
"gsm8k": {
  "exact_match,strict-match": 0.5072,
  "exact_match_stderr,strict-match": 0.0138,
  "exact_match,flexible-extract": 0.5375,
  "exact_match_stderr,flexible-extract": 0.0138
}
```

Task YAML：`lm_eval/tasks/gsm8k/gsm8k-cot.yaml`，关键是 `filter_list` 同时跑 strict 和 flexible 两套 regex：

```
filter_list:
  - name: "strict-match"
    filter:
      - function: "regex"
        regex_pattern: "#### (\\-?[0-9\\.\\,]+)"
      - function: "take_first"
  - name: "flexible-extract"
    filter:
      - function: "regex"
        regex_pattern: "(-?[$0-9.,]{2,})|(-?[0-9]+)"
        group_select: -1
      - function: "take_first"
```

`strict-match` 要求模型严格输出 `#### <answer>` 格式，模型不一定遵守；`flexible-extract` 抓 generation 中最后一个数字，更宽容。两套数字差 2–6 pp 是常态。

29 / 64

坑 1：CoT prompt 模板与原 paper 不同

GSM8K 官方 prompt 是 Wei et al. 2022 CoT paper Table 20 的 8-shot 模板。harness 用的是稍微修改版（少了一个换行），二者差 0.5–1.5 pp。复现 Cobbe et al. 2021 原始数字时建议用 `gsm8k_cot` task variant 而非 `gsm8k`；前者用 paper-aligned prompt，后者用 harness 自己改的简化版。

坑 2：`max_gen_toks` 太短

GSM8K generation 默认 `max_gen_toks=256`，对 7B 模型够用，对 reasoning 模型不够。Llama-3.1-8B base 上 256 token 实测 cut-off 率 < 1%；但若评测 R1-distill 类带 thinking 的模型，必须拉到 32k——这是第 5 章主题。

坑 3：stop sequence 不能是 `\n\n`

如果 stop sequence 配 `["\n\n"]`，模型在 chain-of-thought 中间空一行就停，pass@1 直接掉 15+ pp。harness 默认 stop 是 `["Question:", "</s>"]`，遵循 paper 原文。自定义 stop 时要小心。

与 paper 数字对账

- Llama-3.1-8B-base 这套配置 strict 50.7% / flexible 53.8%，Meta model card 报 8-shot CoT `gsm8k_cot=56.3%`——差距来自 Meta 用了 maj@8 投票。pass@1 应对 pass@1 比。
- GSM8K test set 已确认存在 contamination 与 saturation（Mirzadeh et al. 2024 GSM-Symbolic[^mirzadeh2024gsm-symbolic] 把 GSM8K 的"扰动版"测出 65% 性能塌方）。看到 95%+ 的数字别太兴奋，跑一遍 GSM-Symbolic 或 GSM1k[^zhang2024gsm1k]。

HumanEval：sandbox 与 pass@k 的不可信估计

```
HF_ALLOW_CODE_EVAL=1 lm_eval \
  --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B \
  --tasks humaneval \
  --num_fewshot 0 \
  --confirm_run_unsafe_code \
  --output_path results/llama31-8b/humaneval --log_samples
```

输出：

```
"humaneval": {
  "pass@1,create_test": 0.3475
}
```

Task YAML：`lm_eval/tasks/humaneval/humaneval_instruct.yaml`（instruct）和 `humaneval.yaml`（base）。底层 `lm_eval/tasks/humaneval/utils.py` 调 `openai/human-eval` 的 `execute_check_correctness`，在 subprocess + signal-based timeout 里执行模型 completion 拼接的代码。

坑 1：sandbox 不会替你做隔离

`HF_ALLOW_CODE_EVAL=1` 加 `--confirm_run_unsafe_code` 双开关只是绕过 harness 的安全门——并不代表你跑在沙箱里。原始 `openai/human-eval` 的 `execution.py` 用 `signal.SIGALRM` 杀进程，但模型可能输出 `os.system('rm -rf /')`，subprocess 是用宿主用户跑的。生产实践有两条路：(1) 跑在 Docker container 里（推荐，简单的 `python:3.10-slim` 镜像就够）；(2) 用 nsjail / firejail wrap。EvalPlus 的 `evalplus` CLI 内置 Docker mode（`evalplus.evaluate --samples xxx.jsonl --dataset humaneval`），更友好。

我们组踩过的坑：早期把 `lm_eval` 跑在共享 dev 机器上，一个 bug-y 模型生成 `subprocess.Popen(['kill', '-9', '1'])`，把 host 上其他人的 jupyter session 也搞挂了。**永远 Docker。**

坑 2：pass@k 估计在 n_samples 小的时候有偏

`openai/human-eval` 的 unbiased pass@k 估计公式是：

```
pass@k = E_problem [ 1 - C(n-c, k) / C(n, k) ]
```

n 是采样数、c 是通过数。harness 默认 `n_samples=10` 跑 `pass@1, pass@10`，pass@10 在 n=10 时方差大。要 pass@10 数字稳，n 至少 100。这是 HumanEval saturation 后 paper 数字普遍只报 pass@1 的原因——pass@10 太贵且 n=10 数字 noise 太大。

坑 3：HumanEval 已 saturated，要看 HumanEval+

GPT-4-class 模型在原始 164 题上常 >95% pass@1，没 headroom。Liu et al. 2023 EvalPlus[^liu-2023-evalplus] 把每题 unit test 数从 ~7 扩到 ~560（80× 增强），frontier 模型 pass@1 掉 19–29 pp。lm-eval-harness `humaneval_plus` task 直接用 EvalPlus 的扩展测试集，建议主报 `humaneval_plus pass@1` 而非原版。

BBH：CoT 触发与 23 个子任务的均值陷阱

```
lm_eval --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B,tensor_parallel_size=8 \
  --tasks bbh_cot_fewshot \
  --batch_size auto \
  --output_path results/llama31-8b/bbh --log_samples
```


31 / 64

输出 23 个 subtask + 1 个 `bbh_cot_fewshot` 总分：

```
"bbh_cot_fewshot": {
  "exact_match,custom-extract": 0.6310
},
"bbh_cot_fewshot_boolean_expressions": {
  "exact_match,custom-extract": 0.8480
},
"bbh_cot_fewshot_dyck_languages": {
  "exact_match,custom-extract": 0.2080
}
```

Task 文件夹：`lm_eval/tasks/bbh/`，含 `cot_fewshot/` 和 `zeroshot/` 两个子任务包。CoT prompts 在 `cot-prompts/` 子目录下，每个 task 一个 `.txt` 文件，与原始 `suzgunmirac/BIG-Bench-Hard` repo 同步。

坑 1：CoT vs answer-only 差 10–20 pp 是 BBH 标志

BBH 设计本身是 "CoT 的实证舞台"——23 task 平均，CoT 比 answer-only 高 10–20 pp。harness 的 `bbh_zeroshot` 是 answer-only 协议，`bbh_cot_fewshot` 是 3-shot CoT。报 BBH 数字不指明协议基本无意义。Suzgun et al. 2023[^suzgun2023bbh] 报 PaLM-540B CoT 65.2% vs answer-only 50.4%，差 14.8 pp。

坑 2：23 task 算术平均不加权，但 task size 不等

BBH 23 task 大部分 250 例，但有几个明显少：`boolean_expressions` 250、`web_of_lies` 250、`tracking_shuffled_objects_*` 三 task 各 250、`disambiguation_qa` 250…实际几乎都是 250，但 sample size 在子文件夹里就有差异。harness 默认算术平均（不按题数加权），与 paper 一致——但这意味着一个 `dyck_languages` 跌 20 pp 与 `web_of_lies` 跌 20 pp 同等重要。Debug 时一定要 per-subtask 看，别只看总分。

坑 3：CoT prompt 内嵌的 chain-of-thought 模板必须保留 `\nA:` `Let's think step by step.`

harness 的 `cot-prompts/*.txt` 来自原始 BBH repo。如果你自己改 prompt 想"清理"那行 `Let's think step by step.`，CoT 就不会触发，分数掉到 answer-only baseline。Madaan & Yazdanbakhsh 2023 "Invalid Logic, Equivalent Gains"[^madaan2023invalidcot] 还实测了：把那行换成完全无意义的 `Let's xkcd quantum kettle.` 居然依然能拉 5+ pp ——但前提是触发了 generation 中的"step-by-step"格式。结论是：prompt 触发了"逐步生成"行为本身有用，不是 CoT 内容真。报 BBH 时永远先确认 prompt 是否真的让模型逐步推理（看 `samples_*.jsonl` 抽几条人眼检查）。

已知 ground-truth 错误

BBH GitHub Issue #14 / #15 报 `date_understanding`、`geometric_shapes` 部分 ground-truth 标错，维护方暂未修。复现 paper 数字时这点不影响（大家踩同一个 bug），但你的真实能力数字会被这部分噪声压低 ~0.5–1 pp。Kazemi et al. 2025 BBEH (BIG-Bench Extra Hard)[^bbeh-paper] 已彻底替换。

MBPP：与 HumanEval 同框，但 test 稀疏

```
HF_ALLOW_CODE_EVAL=1 lm_eval \
  --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B \
  --tasks mbpp \
  --confirm_run_unsafe_code \
  --output_path results/llama31-8b/mbpp --log_samples
```

输出：

```
"mbpp": {
  "pass@1,create_test": 0.4060
}
```

Task YAML：`lm_eval/tasks/mbpp/mbpp.yaml`，sanitized 子集是 `mbpp_sanitized` (427 题)，全集是 `mbpp` (974 题，含低质量众包项)。论文主流报的是 sanitized；新人常把 974 题数字写 paper 里，对账时 5–10 pp 对不上。

坑：MBPP 每题 3 个 test，太稀

Liu et al. 2023[^liu-2023-evalplus] 报 MBPP 原版 3 个 test 让 frontier 模型 pass@1 虚高 5–15 pp。MBPP+ (EvalPlus) 把测试扩到 ~100/题，建议主报 `mbppplus`。

Sandbox 注意事项同 HumanEval：Docker 隔离，不要相信 in-process `exec()`。

PIQA：物理常识的 paired log-likelihood

```
lm_eval --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B \
  --tasks piqa --num_fewshot 0 \
  --output_path results/llama31-8b/piqa --log_samples
```

输出：

```
"piqa": {
  "acc,none": 0.8074,
  "acc_norm,none": 0.8123
}
```

Task YAML：`lm_eval/tasks/piqa/piqa.yaml`，与 HellaSwag 同构（paired LL + acc_norm）。Random = 50%。

PIQA 是 2026 年场景中"早已 saturated 但仍报"的代表——前沿模型常 88%+ 而人类 95%，差距已主要由噪声构成。Chang et al. 2025 Global PIQA[^chang2025globalpiqa] 揭示英文 PIQA 与跨语言 PIQA 间有 37 pp gap，纯英文 score 不能代表"物理常识"能力。

ARC-Challenge：25-shot 与小学到中学的科学题

```
lm_eval --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B \
  --tasks arc_challenge --num_fewshot 25 \
  --output_path results/llama31-8b/arc --log_samples
```

输出：

```
"arc_challenge": {
  "acc,none": 0.5538,
  "acc_norm,none": 0.5973
}
```

Task YAML：`lm_eval/tasks/arc/arc_challenge.yaml`。25-shot 是 Open LLM Leaderboard v1 协定；HELM 用 5-shot。两者数字差 2–4 pp。

主报 `acc_norm`（length-normalize），因为 ARC 选项是完整短句，长度差异大。注意 ARC 题部分是 3 选 1 / 5 选 1（少数），harness 处理时 `doc_to_choice` 返回的是 list 长度可变。

TriviaQA：closed-book EM 与 multi-alias 匹配

```
lm_eval --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B \
  --tasks triviaqa --num_fewshot 5 \
  --output_path results/llama31-8b/triviaqa --log_samples
```

输出：

```
"triviaqa": {
  "exact_match,remove_whitespace": 0.6740
}
```

Task YAML：`lm_eval/tasks/triviaqa/default.yaml`。Closed-book 模式：只给问题、模型 generate 答案。EM 用 SQuAD-style normalize（小写、去标点、去 articles `a/an/the`），并对 multiple gold aliases 任一匹配即算对。

坑：closed-book 别忘 stop sequence

TriviaQA closed-book 模型很容易在答出 "John F. Kennedy" 之后继续 generate 一段无关解释。stop sequence 默认 `["\n", "."]`，但 5-shot 模板里 demonstration 答案不带句号，模型有时学到"继续生成"风格。看 `samples_*.jsonl` 抽几条手验 stop 是否正确生效。

Llama-3.1-405B model card 报 5-shot 91.8%[^llama3-card]，但若你用 `--num_fewshot 0` 会掉到 60% 左右。closed-book QA 对 shot 敏感。

NaturalQuestions：closed-book EM + 严重 contamination

```
lm_eval --model vllm \
  --model_args pretrained=meta-llama/Llama-3.1-8B \
  --tasks nq_open --num_fewshot 5 \
  --output_path results/llama31-8b/nq --log_samples
```

输出：

```
"nq_open": {
  "exact_match,remove_whitespace": 0.2412
}
```

Task YAML：`lm_eval/tasks/nq_open/nq_open.yaml`。`nq_open` 是 closed-book short-answer 版（7830 dev），harness 没有 long-answer task。

坑：52% test set 已被 contamination

Llama-3 paper Table 15[^llama3-2024paper] 实测 52% NQ 测试题在 LLama-3 预训练 corpus 中——意味着 closed-book LLM 数字几乎全是"记忆 + 一点泛化"。如果你的模型 NQ EM 比 base 还低，先看是不是 chat template 错配或 stop sequence 截太狠，不要怀疑数据集本身。

GitHub `google-research-datasets/natural-questions` 2026-04-19 已 archive，但 HF mirror `nq_open` 仍可用，数据未变。

综合对账：一个 7B base ckpt 上的预期数字

复现 Llama-3.1-8B base 的经典 benchmark 数字，用本章配置应得到（all base, 0/5-shot 标注同前）：

Benchmark	Shot	Metric	我们	Meta card	Δ
MMLU	5	acc	0.6810	0.68	0.0
HellaSwag	0	acc_norm	0.7920	0.79	0.0
WinoGrande	5	acc	0.7372	0.74	0.0
ARC-C	25	acc_norm	0.5973	0.59	+0.7
GSM8K (strict)	8	exact_match	0.5072	0.563 (maj@8)	-5.6 (协议差)
HumanEval	0	pass@1	0.348	0.331	+1.7
BBH (CoT)	3	exact_match	0.6310	0.62	+1.1

Δ 在 ±2 pp 内基本说明 pipeline 没大问题；GSM8K -5.6 是因为 Meta 报 maj@8 而我们是 pass@1，乘上 1.10–1.12 的 maj 增益就回到一致。这个对账表是每个新 ckpt 跑完的标准 sanity check——任何一项 >5 pp 漂移就要回头查 prompt / stop / tokenizer。

引用

[^biderman2024lessons]: Biderman, S. et al. (2024). *Lessons from the Trenches on Reproducible Evaluation of Language Models*. arXiv:2405.14782.

[^gu2024olmes]: Gu, Y. et al. (2024). *OLMES: A Standard for Language Model Evaluations*. arXiv:2406.08446 (NAACL 2025 Findings).

[^gema2024mmluredux]: Gema, A. P. et al. (2024). *Are We Done with MMLU?* arXiv:2406.04127.

[^chizhov2025goldenswag]: Chizhov, P., Nee, M., Langlais, P.-C., Yamshchikov, I. P. (2025). *What the HellaSwag? On the Validity of Common-Sense Reasoning Benchmarks*. arXiv:2504.07825.

[^surge2024hellabad]: Surge AI. (2024). *HellaSwag or HellaBad? 36% of this popular LLM benchmark contains errors*. https://surgehq.ai/blog/hellaswag-or-hellabad-36-of-this-popular-llm-benchmark-contains-errors.

[^mirzadeh2024gsm-symbolic]: Mirzadeh, I. et al. (2024). *GSM-Symbolic: Understanding the Limitations of Mathematical Reasoning in Large Language Models*. arXiv:2410.05229 (ICLR 2025).

[^zhang2024gsm1k]: Zhang, H. et al. (2024). *A Careful Examination of Large Language Model Performance on Grade School Arithmetic*. arXiv:2405.00332 (NeurIPS 2024 D&B).

[^liu-2023-evalplus]: Liu, J. et al. (2023). *Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation*. arXiv:2305.01210.

[^suzgun2023bbh]: Suzgun, M. et al. (2023). *Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them*. EMNLP Findings 2023. arXiv:2210.09261.

[^madaan2023invalidcot]: Madaan, A., Yazdanbakhsh, A. (2023). *Invalid Logic, Equivalent Gains: The Bizarreness of Reasoning in Language Model Prompting*. arXiv:2307.10573.

[^bbeh-paper]: Kazemi, M. et al. (2025). *BIG-Bench Extra Hard*. arXiv:2502.19187.

[^chang2025globalpiqa]: Chang, T. A. et al. (2025). *Global PIQA: Evaluating Physical Commonsense Reasoning Across 100+ Languages and Cultures*. arXiv:2510.24081.

[^llama3-card]: Meta. *Llama 3.1 Model Card*. https://github.com/meta-llama/llama-models/blob/main/models/llama3_1/MODEL_CARD.md.

[^llama3-2024paper]: Dubey, A. et al. (2024). *The Llama 3 Herd of Models*. arXiv:2407.21783.

[^eleuther-mcq-norm]: EleutherAI Blog. *Multiple Choice Normalization in LM Evaluation*. https://blog.eleuther.ai/multiple-choice-normalization/.

R1 / o1 / o3 之后，跑评测的工程问题与"经典 MMLU/HellaSwag 时代"完全不是一个量级。本章讲六件事：(1) 长 CoT 与 thinking-token 的 `max_tokens` 设置；(2) self-consistency / maj@k 采样的成本曲线；(3) 数学类 benchmark 的 grader 实现（symbolic equivalence、boxed-answer 提取）；(4) 代码类 benchmark 的 rolling cutoff 机制；(5) base model 与 reasoning prompt 的兼容性陷阱；(6) 一份 reasoning-era pretrain ckpt 跑一遍 reasoning suite 的成本估算。读者前置假设同第 4 章，但需要至少 8×H100 80G 来跑 32k context 的批量推理。

长 CoT 与 thinking-token：max_tokens 不再是 2048

reasoning-model 与传统 LLM 的最大工程差异是 thinking-token 流。DeepSeek R1 / o1 / Qwen3-32B Reasoning 等模型在生成 final answer 前会先输出一长串内部推理（在 R1 是用 `<think>...</think>` tag 包裹，在 o1 是 hidden tokens 用户看不到，在 Qwen3 是 prefix `<think>\n` 不带 closing tag），平均长度在 AIME / FrontierMath 难题上常达 8k–32k token，极端时（FrontierMath Tier 3 / 4）超过 64k。

实测 thinking-token 长度分布（DeepSeek R1 在 AIME 2024 上，跑 30 题各 maj@32）：

```
percentile | think_tokens | total_tokens
 5%        |   1,021      |   1,308
50%        |   6,847      |   7,194
75%        |  12,431      |  13,002
90%        |  21,180      |  21,883
95%        |  29,514      |  30,478
99%        |  53,162      |  54,108
max        | 102,847      | 103,884
```

工程含义：默认 `max_tokens=2048` 会在 30% 以上的 AIME 题里直接截断在 `<think>` 中间——model 还没开始写 final answer，整道题被判 0 分。这是 reasoning-era 跑

benchmark 时**所有 reported 数字大滑铁卢的第一原因**。

vLLM 中正确设置：

```
lm_eval \
  --model vllm \
  --model_args "pretrained=deepseek-ai/DeepSeek-R1,tensor_parallel_size=8,max_mo
  --tasks aime_2024 \
  --gen_kwargs "temperature=0.6,top_p=0.95,max_gen_toks=32768,do_sample=True" \
  --output_path results/r1/aime24 --log_samples
```

`max_gen_toks=32768` 是 R1 在 AIME 上的合理默认；FrontierMath Tier 1-3 拉到 65536；Tier 4 必须 131072+（且需要给 vLLM 准备至少 80GB KV cache）。`max_model_len` 是模型自带能力上限（R1 = 128k），别和 `max_gen_toks` 混淆。

DeepSeek 官方 paper 的 AIME 数字用的是 `temperature=0.6, top_p=0.95, maj@64`，不是 greedy；要复现必须用同样采样配置。greedy decoding 在 reasoning model 上分数普遍低 5–10 pp，因为 thinking-token 是高熵 generation。

thinking-token 与 KV cache 的 OOM 陷阱

vLLM `enable_prefix_caching=True`（默认）在 reasoning model 上会咬出难复现的 OOM——thinking-token 都是 unique 的、prefix-share 率近 0，但 cache 仍会反复 alloc/dealloc，碎片化严重。reasoning task 上**显式关掉 prefix cache**（`enable_prefix_caching=False`），用 `paged_attention` 全量管控。

第二个坑：batch size 太大时 KV cache 在 99% 长 generation 完成、1% 还在生成时被卡住——剩下 1% 占着所有 slot。vLLM v0.6.4+ 的 `--max-num-seqs 8` 限制并发数能稳态；新人常默认 256 起跑，结果中位 latency 不变但 P99 爆炸。

Self-consistency / maj@k：成本曲线与置信区间

reasoning era 的"标准报告动作"是 `maj@k`（k 次采样，取多数票答案）。AIME 30 题、maj@64 = 1,920 次推理；R1 平均每次 ~7k token，total ≈ 13.4M token；按 8×H100 80G 部署、throughput ≈ 800 token/s（vLLM bf16，batch=8），wall clock ≈ 4.6 小时。一遍 AIME 2024 + 2025 maj@64 = ~9 小时。

Soohak / FrontierMath / MathArena 都要求 maj@32 或 Avg@3 以稳定数字。算下来一个 R1-size 模型跑全套 reasoning suite（AIME / MATH-500 / GPQA-Diamond / LiveCodeBench v6 / SWE-Bench Pro）一次 ≈ 240 H100-小时，价格约 $480-$720（按 spot $2-3/hr/H100）。

maj@k 的统计置信区间常被低估。AIME 30 题 maj@1 的标准差是 √(p(1-p)/30) ≈ 9.1%（p=0.5）；maj@32 名义上把 p 估计的精度提高 √32 ≈ 5.7 倍，但 effective sample size 还是 30（你只有 30 题），CI ≈ ±5 pp 是天花板。AIME 2025 上 R1 报 79.8% 与 79.2% 在 CI 内

38 / 64

不可分。MathArena 论文[^dekoninck2026matharena] 显式给 bootstrap CI，这是 reasoning era 该有的报告规范——单一数字不附 CI 没意义。

数学 grader：boxed-answer 提取与 SymPy normalize

MATH / MATH-500 / AIME / FrontierMath 的评分骨架是相同的："抽取 `\boxed{...}` 内表达式 → SymPy normalize → 与 gold 比较"。代码长这样：

```
import re
from sympy import sympify, simplify, Rational

def extract_boxed(text: str) -> str | None:
    # 抓最后一个 \boxed{...}, 处理嵌套大括号
    matches = []
    for m in re.finditer(r"\\boxed\{", text):
        start = m.end()
        depth, i = 1, start
        while i < len(text) and depth > 0:
            if text[i] == "{": depth += 1
            elif text[i] == "}": depth -= 1
            i += 1
        if depth == 0:
            matches.append(text[start:i-1])
    return matches[-1] if matches else None

def math_equivalent(pred: str, gold: str) -> bool:
    if pred is None: return False
    # 先做字符串规范化
    pred_norm = pred.replace(" ", "").replace("\\$", "").replace(",", "")
    gold_norm = gold.replace(" ", "").replace(",", "")
    if pred_norm == gold_norm: return True
    # 再走 SymPy
    try:
        p, g = sympify(pred_norm), sympify(gold_norm)
        return simplify(p - g) == 0
    except Exception:
        return False
```

这套代码看着简单，实际 corner case 多到一只手数不完。常见 fail mode：

1. **嵌套大括号**：`\boxed{\frac{1}{2}}` 简单正则 `\\boxed\{(.+?)\}` 会抓 `\frac{1`，必须手写 depth counter。
2. **LaTeX 与 SymPy 语法差异**：`\frac{1}{2}` SymPy 不认，要先 `latex2sympy2` 转换或自己写 `\frac` → `(1/2)` 替换。
3. **分数 vs 小数**：`1/2` 与 `0.5` SymPy `simplify(1/2 - 0.5) == 0`，OK；但 `\sqrt{2}` 与 `1.4142...` 不等价（容差问题）。

39 / 64

4. **多答案题**：MATH 极少数题 ground truth 是 `(2, 3)` 这种 tuple，pred 写 `(3, 2)` 算不算对？官方答案是看题——有些题顺序重要、有些不重要。这里 OLMES 与 lm-eval-harness 的实现都默认顺序敏感，与 MATH 原 paper 一致。

lm-evaluation-harness 的实现在 `lm_eval/tasks/hendrycks_math/utils.py`，核心 `is_equiv()` 函数 ~200 行，覆盖了上面大部分 case。MATH-500 task `lm_eval/tasks/hendrycks_math/hendrycks_math500.yaml` 直接复用同 utils。

AIME 答案是 0–999 整数，grader 简化：

```
def aime_check(pred: str, gold: int) -> bool:
    boxed = extract_boxed(pred)
    if boxed is None:
        # fallback: 抓 generation 最后的整数
        nums = re.findall(r"-?\d+", pred)
        if not nums: return False
        boxed = nums[-1]
    try:
        return int(boxed.strip()) % 1000 == gold % 1000
    except ValueError:
        return False
```

`% 1000` 是因为 AIME 答案保证 0–999，模型偶尔写 `\boxed{1234}` 实际想表达的是 234（mod 1000）；MathArena 的 grader 走这条路。

FrontierMath 与 MathArena 的 grader 差异

FrontierMath 每题有作者预设的 verification function——SymPy 检验 / 数值检验 / 结构匹配——题级 grader 而非通用 grader。所有评测必须在 Epoch AI 自己的 server 上跑（题在 hold-out set 中），普通用户没法本地跑。2026-05-11 Epoch AI 自己发现 ~1/3 题 ground-truth 有 fatal errors[^epoch-tier4]，leaderboard 临时 frozen——这暴露了非公开 grader 的 governance 风险。

MathArena[^dekoninck2026matharena] 走另一条路：grader 全公开，generation + LLM-as-judge（Gemini-3-Flash 作 judge）+ 人工抽检 false positive。所有题、所有 generation、所有 judge log 都在 `huggingface.co/MathArena` 公开。CLI：

```
uv run scripts/run.py --comp arxivmath_2026_04 --models gpt-5 --n_runs 4
```

ArXivMath 每月 ~40 题，n_runs=4 即 ~160 次推理；Apex set 60 题、USAMO 6 题、AIME 2026 30 题。MathArena 月卷数字不可单看——CI ±5% 量级。

GPQA-Diamond：MCQ 的 reasoning 协议

GPQA-Diamond 198 题，4 选 1。reasoning model 用 zero-shot CoT + answer-letter 提取：

```
lm_eval --model vllm \
  --model_args "pretrained=Qwen/Qwen3-32B-Instruct,tensor_parallel_size=8,max_mo
  --tasks gpqa_diamond_cot_n_shot \
  --num_fewshot 0 \
  --gen_kwargs "temperature=0.7,max_gen_toks=16384" \
  --output_path results/qwen3-32b/gpqa --log_samples
```

Task YAML：`lm_eval/tasks/gpqa/cot_n_shot/gpqa_diamond_cot_n_shot.yaml`（diamond 实例），共享 template `_gpqa_cot_n_shot_yaml`。`filter_list` 抓 generation 中最后的 `(A) | (B) | (C) | (D)` 形式。

坑：模型经常输出 `the answer is option C` 而不是 `\boxed{C}` 或 `(C)`。OpenAI o-series 的报告里 GPQA 用的是 maj@N，把每次 sample 抽到的 letter 投票——这意味着 generation 里 letter 一定要能被 robust 抽出。filter regex 写不严格，原本 87% 的模型测出来掉到 78%（被 ambiguous extraction 拉低）。

robust answer extractor 经验值：

```
# 按优先级试三套抽取
def extract_gpqa_letter(text: str) -> str | None:
    # 1) \boxed{X}
    m = re.search(r"\\boxed\{\s*([ABCD])\s*\}", text)
    if m: return m.group(1)
    # 2) "answer is (X)" / "answer is X"
    m = re.findall(r"(?:answer|option)\s+(?:is\s+)?\(?([ABCD])\)?", text, re.I)
    if m: return m[-1].upper()
    # 3) generation 末尾出现的 standalone letter
    m = re.findall(r"\b([ABCD])\b", text)
    if m: return m[-1]
    return None
```

三层 fallback 比单一 `\\boxed{}` 抽出率高 8-12 pp（实测 R1-Distill-Llama-70B 上 79% → 89%）。lm-evaluation-harness 在 `lm_eval/tasks/gpqa/cot_n_shot/utils.py` 用类似策略。

198 题 binomial CI：±2 pp 是噪声地板。frontier model 在 GPQA-Diamond 上 92% 与 94% 不可分。Epoch AI[^gpqa-epoch] 估算 ~10% 题目本身有效性可疑，"真" ceiling 在 90% 上下。

LiveCodeBench：月更窗口与 release_version 过滤

LiveCodeBench (LCB)[^jain-2024-livecodebench] 的核心机制：每月从 LeetCode / AtCoder / Codeforces 拉新题，按 release date 标签。每次评测必须指定 `--release_version` 和（可选）`--start_date / --end_date`，把模型只测在 cutoff 之后发布的题上。

```
# LCB 自家 harness，不在 lm-eval-harness 主线
git clone https://github.com/LiveCodeBench/LiveCodeBench
cd LiveCodeBench
python -m lcb_runner.runner.main \
  --model deepseek-ai/DeepSeek-R1 \
  --scenario codegeneration \
  --release_version v6 \
  --start_date 2025-01-01 \
  --end_date 2025-04-30 \
  --evaluate \
  --temperature 0.2 \
  --n 1 \
  --max_tokens 32768
```

v6 是 2026-05 当前 stable release，含 2023-05 至 2025-04 共 1,055 题。"contamination-free" 评测对一个 2024-12 cutoff 的模型应该用 `start_date=2025-01-01`，仅评测 ~150 题。问题：150 题的 binomial CI 是 ±4 pp，frontier 区分度有限。LCB-Pro[^livecodebench-pro-2025] 是 IOI/ICPC 奖牌得主重新标注的更难版本，2025-06 发布；frontier model 在 medium 题只有 53%、hard 题 0%——这是 LCB v6 已 saturated 的实证。

Sandbox：Docker 是必须

LCB 用 `containerd-snapshotter` + `python:3.10-slim` Docker container 跑 generation。和 HumanEval 一样不能在主机直接跑——竞赛题里 `while True: pass` 是常见 fail，没 timeout 会卡死 worker。LCB 自带 30 秒 hard timeout + 4GB memory cap。生产环境推荐用 `--num_process_evaluate 32` 起 32 个并行 sandbox，跑 1000 题 ≈ 25 分钟。

LCB 还有 `test_output_prediction` 与 `code_execution` 两个 scenario。前者让模型不跑代码预测输出，后者反过来给函数 + 输入预测函数行为。两者都不需要 sandbox（只是 generation + EM）。

SWE-Bench Pro：repo-level 与 Docker pinned image

```
# SWE-Bench Pro 没有公开本地 harness：评测通过 Scale AI leaderboard 提交 prediction
# 提交 JSONL 格式与 SWE-bench Verified 兼容（instance_id + model_patch + model_name
# Scale AI 内部跑 Docker pinned image，返回 pass/fail breakdown
# Public 731 题可在 HuggingFace `ScaleAI/swe-bench-pro` dataset 查看 instance_ids
# Private 276 题 + Held-out 858 题 永远本地拿不到（API submission only）
```

每个 `instance_id` 对应一个 Scale AI 预打包 Docker image（约 30 个 image，每个 800MB-2GB），含该 repo 的 pre-fix state 与 test runner。harness apply patch → run test suite → compare against expected FAIL_TO_PASS / PASS_TO_PASS markers。Public 731 题、Private 276 题（API only）、Held-out 858 题（Scale AI 内部周期性 refresh）。

SWE-Bench Pro vs SWE-bench Verified：工程差异

Verified 500 题在 12 个 repo（绝大多数 Python），SWE-Bench Pro 1865 题在 41 个 repo、4 种语言。工程上 3 个明显差异：

1. **多语言 runtime**：Verified Python-only，单 image 容易共用；Pro 含 Go / TS / JS，每语言一套 toolchain，Docker image 总大小从 Verified ~15GB 涨到 Pro ~85GB。本地存盘前先看磁盘。
2. **patch 大小**：Verified avg 1-2 LOC change in 1 file；Pro avg 107 LOC across 4.1 files。这意味着模型必须能在 ≥4 个文件间保持一致的修改——对 reasoning model 的 long-horizon planning 是真实考验。MorphLLM[^morphllm-2026-pro-gap] 测出 frontier model 在 Pro 比 Verified 掉 30-45 pp，主要在 multi-file 错位。
3. **私有集**：Pro Private 276 题在 Scale 内部代码库（startup 工程师未公开的代码），用户提交 prediction，Scale 跑评测返回数字——你看不到题。这套 governance 类似 FrontierMath 但比 FrontierMath 多了"商业 NDA"维度，Scale 经济利益与 benchmark independence 之间的张力比 Epoch AI 更直接。

LiveCodeBench 与 SWE-Bench Pro 配对是 2026 frontier code LM 的标配：前者测单文件算法（saturated 信号弱但 cheap），后者测 repo-level（区分度强但 expensive，~$3-5/task with frontier model agents）。

Base model 与 reasoning prompt 的兼容性

reasoning benchmark 几乎全部围绕 instruct/reasoning 模型设计，base ckpt 跑这些 benchmark 通常没意义但工程问题更隐蔽：

- **`<think>` tag 不会自动出现**：DeepSeek R1 是从 R1-Zero 强化学习得来的，base ckpt（R1-Zero 之前）不会自发产 `<think>`。你强制 prompt 里加 `<think>\n` 让模型续写，得到的也只是普通 next-token 续写，不是真正的 chain-of-thought。
- **chat template 不能开**：base ckpt 跑 GPQA-Diamond 想测 "5-shot CoT" 协议，必须 `--apply_chat_template false`，否则 base 模型看到 `<|im_start|>system\n` 整体崩溃。
- **temperature 必须高**：base ckpt 在 reasoning 类 task 上 greedy decoding 会落入"answer-only"陷阱（不生成 chain），需 `temperature=0.8` + 长 `max_tokens` 强制延展 generation。

base ckpt 跑 reasoning benchmark 唯一合理的场景是 in-training-probing[^liu2026probing]：把 pretrain 中间 checkpoint 当 indicator，看 GSM8K / MATH 的趋势线是否正在改善。绝对数字不重要、变化方向重要。

Thinking-token 与 generation-only 区分

一个 reasoning era 评测易错的地方：报告分数时把 thinking-token 也算进 "generation length"，但 thinking 内容不应进 answer extraction。R1 输出格式：

```
<think>
Let me think carefully. The problem asks for...
[5000 tokens of internal reasoning]
</think>

The answer is \boxed{42}.
```

正确做法：grader 应只看 `</think>` 之后内容（"the answer is" 那段），thinking 内容仅记录但不参与分数。错误做法：把整段 generation 喂 regex 抽 `\boxed{}` ——thinking 里 model 会试好几个候选答案 "Let me check: \boxed{43}... wait, that's wrong... \boxed{42}"，regex 抽到 `42` 还算对，但若 model 在 thinking 里写 `\boxed{40}` 在收尾忘了重新 `\boxed`，最终 fallback 抓到 `40` 就错了。

实现上：

```
def split_thinking(text: str) -> tuple[str, str]:
    if "</think>" in text:
        think, answer = text.split("</think>", 1)
        return think + "</think>", answer
    # 没有 </think>: 假设整段都是 thinking（被截断了）
    return text, ""
```

vLLM 0.6.6+ 提供 `reasoning_parser="deepseek_r1"` 选项自动切分，但若你跑的是别的 reasoning model（Qwen3、Kimi-k1.5），需自己实现。MathArena 的 `scripts/run.py` 在 `lib/parsing.py` 里维护一组各家 reasoning model 的 thinking 分隔符。

MathArena / LemmaBench / Soohak：API-only 与 hold-out

新一代 reasoning benchmark 越来越倾向 API-only / hold-out 模式：

- **MathArena**：grader 全公开，但题可选 mine 自 arXiv（公开）或 USAMO（公开）。本地 reproducible。
- **LemmaBench**[^peyronnet2026lemmabench]：题源 arXiv 公开，但 self-containment filter + proof LLM-judge 都需要 API access；理论可 local 跑但成本高（677 lemma × 1 generator + 1 judge call）。
- **Soohak**[^son2026soohak]：题 partially private，dataset 计划 2026 年底公开。当前只能向作者申请 evaluation。Refusal subset 99 题是"病态题"——模型应回答 "this problem has issue X" 而非给数字答案。grader 是 LLM-judge 抽取 + 人工抽检。
- **FrontierMath**：题完全私有（Tier 4 中 48/50 为 private），全部 evaluation 由 Epoch AI 跑，self-reported 数字。

工程上这意味着 reasoning-era 评测的"我能本地跑通"门槛在快速上升。lm-eval-harness 主线只覆盖到 AIME / MATH-500 / GPQA-Diamond / LiveCodeBench（部分）。Soohak / FrontierMath / MathArena 都是各自团队自家 CLI。一个负责任的 release report 至少要 (a)

自家 lm-eval-harness 跑可复现部分；(b) 提交 evaluation 申请到至少 1 个 hold-out benchmark；(c) 明确标注 self-reported vs 第三方验证。

一份完整成本估算（参考）

7B reasoning-distill 模型跑全套 reasoning suite（除 FrontierMath / Soohak 之外的本地可跑部分）的成本，8×H100 80G、vLLM bf16、`max_gen_toks=32k`、平均 thinking ~5k token：

| Benchmark | 题数 | k (maj@k) | 总 sample | wall clock | GPU-hr | |---|---|---|---|---|---| | AIME 2024+2025 | 60 | 32 | 1,920 | ~6 h | 48 | | MATH-500 | 500 | 4 | 2,000 | ~3 h | 24 | | GPQA-Diamond | 198 | 4 | 792 | ~1.5 h | 12 | | LiveCodeBench v6 (post-cutoff 150 题) | 150 | 1 | 150 | ~30 min | 4 | | BBEH (24 task × ~100 题) | ~2400 | 1 | 2,400 | ~2 h | 16 | | MathArena ArXivMath 月卷 | ~40 | 4 | 160 | ~30 min | 4 | | LemmaBench | 677 | 1 + judge | ~1,400 | ~3 h | 24 | | **小计** | — | — | ~8,800 | ~16.5 h | **~132** | | SWE-Bench Pro Public | 731 | 1 | 731 | ~12 h (agent loop) | 96 | | **含 SWE-Pro 合计** | — | — | ~9,500 | ~28 h | **~228** |

按 spot $2.5/hr/H100：本地可跑 reasoning suite 一次 $570，含 SWE-Pro 一次 ~$1,140。这只是一次 pretrain ckpt 的 final eval；training mid-run 不可能这么跑。

正确的 cadence 是：(1) 每 100k step 跑 `MATH-500 + GPQA-Diamond + AIME` 三件套 maj@1 quick check（≈ 1 GPU-hr）；(2) 每 milestone 跑全套 maj@k；(3) Release-only 再跑 SWE-Pro / FrontierMath / Soohak 的 hold-out 申请评测。把全套用作 mid-run monitor 是预算自杀。

引用

[^dekoninck2026matharena]: Dekoninck, J. et al. (2026). *Beyond Benchmarks: MathArena as an Evaluation Platform for Mathematics with LLMs*. arXiv:2605.00674.

[^epoch-tier4]: Epoch AI. (2026). *FrontierMath Tier 4 benchmark page*. https://epoch.ai/benchmarks/frontiermath-tier-4.

[^gpqa-epoch]: Burnham, G. (2025). *GPQA Diamond: What's Left?* Epoch AI Gradient Updates. https://epoch.ai/gradient-updates/gpqa-diamond-whats-left.

[^jain-2024-livecodebench]: Jain, N. et al. (2024). *LiveCodeBench: Holistic and Contamination Free Evaluation of LLMs for Code*. arXiv:2403.07974.

[^livecodebench-pro-2025]: LiveCodeBench Pro Team. (2025). *LiveCodeBench Pro: How Do Olympiad Medalists Judge LLMs in Competitive Programming?* arXiv:2506.11928.

[^morphllm-2026-pro-gap]: MorphLLM. (2026). *SWE-Bench Pro vs Verified Contamination Gap*. https://www.morphllm.com/swe-bench-pro.

[^peyronnet2026lemmabench]: Peyronnet, A., Gloeckle, F., Hayat, A. (2026). *LemmaBench: A Live, Research-Level Benchmark to Evaluate LLM Capabilities in Mathematics*.

45 / 64

arXiv:2602.24173.

[^son2026soohak]: Son, G., Kim, S., Arnett, C. et al. (2026). *Soohak: A Mathematician-Curated Benchmark for Evaluating Research-level Math Capabilities of LLMs*. arXiv:2605.09063.

[^liu2026probing]: Liu, Z. et al. (2026). *Fast and Accurate Probing of In-Training LLMs' Downstream Performances*. arXiv:2604.01025.

第 6 章 · Live / contamination-resistant 跑法

写过 lm-evaluation-harness 的工程师对 MMLU、HellaSwag 这套 static benchmark 的 pipeline 都很熟：拉数据集 → tokenize → batch inference → 算 EM/loglikelihood → 报数。但 LiveBench / LiveCodeBench / MixEval-Hard / Arena-Hard 这一档不一样——它们的 ground truth 每月在动，提交协议带远端 judge，本地复现没有"绝对正确"的标签。本章把这条路线的工程实操拆开来讲，含月更机制、submission API、自建 contamination detection 的代码，最后讨论 Stage 0.5 frontier scan 看到的 2026 新方法（CCV / dataset watermarking / soft contamination / sealed exam / judge reliability）在生产环境里的可行性。

6.1 LiveBench 的月更引擎与本地复现

LiveBench 的核心承诺是"每月 +50 新题"，github.com/LiveBench/LiveBench 的 `livebench/` 目录是数据 + grader 的合并仓库。关键路径：

- `livebench/process_results/` 装 6 个 category 的 grader（math / coding / reasoning / language / instruction_following / data_analysis），每个文件结构一致：`def process_results(question, ground_truth, llm_answer) -> float`，return 0–1 分；
- `livebench/model/models.py` 注册可评测模型，按 release date 锁版本；
- `livebench/scripts/run_livebench.sh` 是入口脚本，调用 `livebench.gen_model_answer` → `livebench.gen_ground_truth_judgment` → `livebench.show_livebench_result`。

跑一次的最小命令：

```
git clone https://github.com/LiveBench/LiveBench
cd LiveBench && pip install -e .

# 用 vllm 跑自家 ckpt
python -m livebench.gen_model_answer \
  --model-path /ckpts/my-model-2026-05 \
  --model-id my-model-v0.5 \
  --bench-name live_bench/coding/coding_completion \
  --livebench-release-option 2025-04-25 \
  --num-gpus-total 8 --max-tokens 4096

# 跑 grader
python -m livebench.gen_ground_truth_judgment \
  --bench-name live_bench --livebench-release-option 2025-04-25 \
  --model-list my-model-v0.5

python -m livebench.show_livebench_result --model-list my-model-v0.5
```

`--livebench-release-option` 是 LiveBench 工程上最反直觉的字段：你以为 release tag 像 git tag 一样只锁数据快照，实际它同时锁两件事——题目集合 + grader 的行为。比如 `2024-11-25` 仍然在 repo 里能跑，但 grader 对 coding tasks 的 unit test set 已经被 2025-04 release 替换过；不指定 release 默认拉最新月度，会出现"上次跑 0.71，重新拉一次同 ckpt 跑 0.68"的不可复现现象。工程上的规则是：**任何 paper claim 都必须钉住 release option**，并在 method section 写明 `livebench-2025-04-25` 这类完整字符串。

LiveBench 的 coding category 用 `gen_ground_truth_judgment` 把 LLM 输出与 unit test 跑 subprocess 比对，sandbox 实现在 `livebench/process_results/coding/utils.py`，本质是 `subprocess.run(..., timeout=10)` + 单独 `tempfile.TemporaryDirectory`，没有容器隔离。给自家集群跑就要小心：万一被测模型生成 `os.system("rm -rf /")` 之类的代码，会直接打到 host。建议至少包一层 `firejail --quiet --net=none --private` 或 `runuser -u nobody`，工业级跑法用 `docker run --rm --network none --cpus 1 --memory 2g python:3.11`。

旧题退役机制是 LiveBench 维护的关键细节：题目在 `livebench/data/<category>/<task>/question.jsonl` 中保留 6–12 个月的 release version，每条 question 带 `release_date`。当某题被 ≥3 个 frontier model 答对到 100% 时，下个月 cycle 会被替换。这就引出一个工程含义：**LiveBench 上的 "trend" 是在变化的 substrate 上画的**——你横向比 2024-11 vs 2025-04 的 GPT-4 score，本质是不同题目的 score；如果不分 release 报数，等于把 noise 当 signal。LiveBench paper 自己也承认月度间的 std 大约 0.36（在 0–100 scale 上）`[livebench-paper]`。

6.2 LiveCodeBench 的 cutoff 管理

LiveCodeBench (github.com/LiveCodeBench/LiveCodeBench) 的设计目标不一样——它不是月更题，而是按时间窗口切代码题，强制 model 的训练 cutoff 早于题目日期。

47 / 64

`lcb_runner/runner/main.py` 的入口在 `--start_date YYYY-MM-DD --end_date YYYY-MM-DD` 上：

```
python -m lcb_runner.runner.main \
  --model gpt-4o-2024-08-06 \
  --scenario codegeneration \
  --start_date 2024-09-01 --end_date 2024-12-31 \
  --release_version release_v6
```

这套机制对 pretrain 团队很关键：你的 base ckpt cutoff 是 2025-03，那 LiveCodeBench v6 (2025-04 release，含 2024-09 → 2025-03 题) 仍然有 contamination 风险；要拿 contamination-clean 数字，必须 `--start_date 2025-04-01` 之后的子集。LiveCodeBench 的 `lcb_runner/utils/extraction_utils.py` 里写明每题带 `contest_date` 字段，grader 在 `lcb_runner/evaluation/compute_code_generation_metrics.py` 跑 unit test 用 subprocess + 7 秒超时。

这里有一个反复出现的工程陷阱：LiveCodeBench 用 LeetCode / AtCoder / CodeForces 三家爬下来的题，但**这三家平台自己的题库会被 web crawler 全量抓走**。哪怕你的 cutoff 严格 < 比赛日期，model 也可能见过同题的"博客解题报告"——这正是 soft contamination paper (arxiv:2602.12413) 实测的现象：CodeForces 78% 题目在 Olmo3 训练 corpus 中有语义重复 `[soft-contam-2026]`。LiveCodeBench 的 cutoff 只能挡住"题目本身"，挡不住"题目的二手讲解"。工程上的应对：(a) 跑 self-built ngram filter（下一节）筛掉训练数据；(b) 在 paper 里诚实写 "after-cutoff subset" 数字的 caveat。

6.3 MixEval-Hard 的 wild-query 复现

MixEval (github.com/JinjieNi/MixEval) 是另一种 contamination-mitigation 思路：不出新题，而是从真实 web user queries 中按 embedding 相似度挑出与 14 个 source benchmark（MMLU / GSM8K / TriviaQA / ...）最相关的题，构造 mix 后取 hard subset。`mix_eval/eval.py` 入口：

```
python -m mix_eval.eval \
  --model_name local_chat \
  --model_path /ckpts/my-model \
  --benchmark mixeval_hard \
  --version 2024-08-11 \
  --batch_size 32 --output_dir results/
```

要注意：MixEval 的 `--version` 是 dynamic bench tag，README 直到 2026-05-25 仍然只挂着 `2024-08-11`——也就是说自 2024-08 之后**没有公开新版本** `[mixeval-repo]`。这与 paper 设计意图（"周期性 refresh，每次 ~85% query unique"）有偏差，工程团队拿 MixEval 当 anti-contamination 主榜要谨慎。社区已经把 MixEval-Hard 视为"NeurIPS 2024 时期 Arena 替代品 baseline"，而非 active dynamic benchmark。

MixEval 的 judge 配置在 `mix_eval/utils/judge_open_ended.py`——free-form 题用 GPT-4 turbo 当 judge，judge prompt 在 `mix_eval/prompts/judge_freeform_for_models.py`。要本地复现就得提供 `OPENAI_API_KEY` 或重定向到 vLLM-hosted Claude/Gemini。MCQ 题不需 judge，走 EM。我们在自家集群跑过一次 MixEval-Hard，~1K 题，GPT-4-turbo judge cost 大约 $25（与 paper 报告的 6% MMLU cost 一致）。

6.4 Arena-Hard v2.0 的 Bradley-Terry pipeline

Arena-Hard-Auto (github.com/lmarena/arena-hard-auto) v2.0 (2025-04-23) 把流程改成 ensemble judge + style control。命令行：

```
python gen_answer.py --setting-file config/gen_answer_config.yaml \
  --endpoint-file config/api_config.yaml

python gen_judgment.py --setting-file config/judge_config.yaml
python show_result.py --judge-name gpt-4.1-2025-04-14 --style-control
```

`config/judge_config.yaml` 里 `pairwise: true` + `bench_name: arena-hard-v2.0` 是关键。Style control 的实现在 `utils/eval_style_control.py`：把 token length + markdown 元素抽出来作为协变量，跑加权回归。Bradley-Terry 聚合在 `utils/elo_analysis.py`，bootstrap 100 次给 95% CI。

工程实操要注意三个坑：(1) **judge model 必须固定**——v2.0 默认是 GPT-4.1 + Gemini 2.5 ensemble，换 judge 数字就不可比；(2) **baseline model 改了**——v1 用 GPT-4-0314，v2 用 GPT-4.1，旧 paper 的 80% win-rate 在 v2 跑出来可能只 50%，因为 baseline 变强了；(3) **judge prompt v2 显式要求评估者 "ignore length and style"**，但论文实测 verbosity bias 仍未根除 `[verbosity-bias]`。LMSYS 自己也在 blog 里承认 self-preference / verbosity / position 三类偏置 `[arena-hard-blog]`。

6.5 自建 contamination detection：能落地的工具链

抛开 LiveBench / MixEval 这类外部 benchmark，pretrain 团队往往需要自查自己的训练数据是否污染了一份内部 eval set。三种工具可立刻落地：

(1) n-gram overlap 经典法。lm-evaluation-harness 的 `lm_eval/decontamination/` 目录给出 13-gram filter：

```
# 简化版，参考 lm_eval/decontamination/janitor.py
from collections import defaultdict

def ngram_overlap(test_set, train_corpus, n=13):
    test_ngrams = set()
    for ex in test_set:
        toks = ex.split()
        for i in range(len(toks) - n + 1):
            test_ngrams.add(" ".join(toks[i:i+n]))
    hits = defaultdict(list)
    for doc_id, doc in enumerate(train_corpus):
        toks = doc.split()
        for i in range(len(toks) - n + 1):
            ng = " ".join(toks[i:i+n])
            if ng in test_ngrams:
                hits[ng].append(doc_id)
    return hits
```

这套方法的工程优势：fast（grep 级别）、reproducible、零 ML 依赖。缺点：13-gram 滑窗对 paraphrase 几乎完全失效。AI2 在 Olmo3 训练 pipeline 中跑过 13-gram filter（参 OLMES 文档），但 Spiesberger et al. 2026 实测：在 Olmo3 训练 corpus 中**CodeForces 78% / ZebraLogic 50% 的题目存在语义重复**，这些重复全部漏过了 ngram filter `[soft-contam-2026]`。论文里跑的实验是：把已知 benchmark 题的"语义改写版"加入 fine-tune 数据，模型在原题上分数大涨，**严格 held-out 题分数也涨**——说明 ngram-clean 不等于 contamination-free。

(2) canary token 主动 inject。Carlini 2019 的经典方法：在你公开的 benchmark 中嵌入一个独特字符串（如 random 32-char hex token），等 6-12 个月看哪些公开模型能 complete 这个 token：

```
import secrets
canary = f"CANARY_{secrets.token_hex(16)}"
# 把 canary 插入 benchmark 的 example metadata，公开 release
# 等 N 月后跑 generation: "Please repeat the canary token: CANARY_"
# 若模型能正确续写，说明它训练数据包含了这份 benchmark
```

这套思路简单但有两个问题：(a) canary 容易被人工 dedup script 过滤掉（特别是看着像 hash 的 string）；(b) 没有 false-positive rate 的理论控制。Dataset Watermarking for Closed LLMs (arxiv:2605.06865) 给出 provable 改进：不依赖单一字符串，而是通过 rephrasing 提升 dataset 中随机词对的共现频率，训练后用统计假设检验在 model output 上验证。abstract 给出的数字是当 watermarked dataset 占 fine-tuning tokens ~1% 时仍可 p<0.01 检出 `[closed-watermark-2026]`。工程角度看，这套方法**还停在 paper 阶段**——repo URL 在 arxiv abstract 页未公开，pretrain 团队当前能用的还是经典 canary。

(3) 黑盒行为探针 (CCV)。Cross-Context Verification (arxiv:2603.21454) 是 2026 最新方向：让 model 在 N 个独立 session 中解同一道题，测多次解法的 diversity。Contaminated

模型 → 解法高度一致；clean 模型 → 自然 variation。在 9 道 SWE-bench Verified × 5 trial 上实现 perfect separation (Mann-Whitney U=0, p≈0.012, r=1.0) `[ccv-2026]`。

工程上模拟一下：

```
# 黑盒 CCV，伪代码
import scipy.stats

def ccv_probe(model_api, problem, n_trials=5):
    responses = []
    for _ in range(n_trials):
        # 每次开新 session，禁用 caching
        r = model_api.complete(problem, temperature=0, fresh_session=True)
        responses.append(r)
    # diversity 度量：可以是 pairwise edit distance、AST diff、变量命名差异等
    diversity = compute_diversity(responses)
    return diversity

clean_set_div = [ccv_probe(model, p) for p in known_clean_problems]
test_set_div = [ccv_probe(model, p) for p in suspected_problems]
u, p_val = scipy.stats.mannwhitneyu(test_set_div, clean_set_div, alternative='le
```

CCV 的工程成本是 N 倍推理预算，对大规模 benchmark 全量检测不现实，更适合"诊断已发布闭源模型对某个 benchmark 是否污染"这类 forensic 用法。**该方法在 abstract 中声称 release code/data 但 arxiv 页未给具体 URL**——和 Closed-LLM Watermarking 类似，目前还是 paper-only。

6.6 Stage 0.5 frontier scan 中 contamination methods 的工程可行性

我们在 Stage 0.5 frontier scan 里捞到 5 个 2026 新 contamination 方向 + 1 个 sealed-exam 协议 + 1 个 judge reliability 工具，逐一从工程视角评估：

方法	arxiv	工程实操可行性	障碍
CCV / HCCA	2603.21454	可自实现，~50 行	repo 未公开，N 倍推理成本，仅 9 题验证
Dataset Watermarking (closed-LLM)	2605.06865	dataset owner 才能用	无 reference impl，attacker 可 paraphrase 干扰
Soft Contamination (semantic dupes)	2602.12413	需 TB 级 embed 全 corpus	无 turnkey tool、阈值主观
LLM Olympiad (sealed-exam)	2603.23292	协议倡议，无代码	需赛事组织方 + 独立 infra
Judge Reliability Harness	2603.05399	**已开源** github.com/RANDCorporation/judge-reliability-harness	仅 4 benchmark × 4 judge 验证
Provable Joint Decontamination (JECS)	2605.21543	conformal procedure，未释 impl	多模型联合判定，单一团队难复现

唯一**今天就能落地**的是 Judge Reliability Harness——RAND Corporation 维护 active 的 repo，给你一个 judge model + benchmark，自动生成 perturbation 测试套件（formatting / paraphrasing / verbosity / label flipping 四种），输出 reliability matrix。装法：

51 / 64

```
git clone https://github.com/RANDCorporation/judge-reliability-harness
cd judge-reliability-harness && pip install -e .
python -m judge_reliability.run \
  --judge-model anthropic/claude-3.5-sonnet \
  --benchmark harmbench \
  --perturbations formatting paraphrasing verbosity label_flip \
  --output reports/claude-judge-harmbench.json
```

RAND 团队的 paper 在 4 个 SOTA judge × 4 benchmark (safety / persuasion / misuse / agentic) 上 stress test 后的结论："No judge that we evaluated is uniformly reliable across benchmarks" `[judge-reliab-2026]`。意味着如果你在 production 用 LLM-as-judge 跑 Arena-Hard 风格评测，**上线前必须 perturbation test**——judge model A 在 safety benchmark 上稳，在 persuasion benchmark 上可能就会 flip。

剩下五个方法的工程现状：CCV 与 Dataset Watermarking 是"理论可用、工业等 repo"；Soft Contamination 是 framing paper，给出"为什么 n-gram 不够"的结论但没给 turnkey tool；LLM Olympiad 是社区组织级倡议，paper 里也没 demo run，作者明确说"我们不提供 reference implementation"；JECS 是 conformal procedure 的理论延伸，需要多个 model 提供方协作做 joint decontamination，单一团队没法跑。

6.7 工程总结：刷 live benchmark 该有的纪律

写到这里，给读者一份在 live / contamination-resistant 这条线上跑 eval 的最小纪律清单：

1. **release tag 钉死**。LiveBench `--livebench-release-option 2025-04-25`、LiveCodeBench `--release_version release_v6 --start_date ... --end_date ...`、MixEval `--version 2024-08-11`、Arena-Hard `--bench-name arena-hard-v2.0`。任何 paper / 内部 report 都要在 method 段写完整版本字符串。
2. **judge model 钉死**。Arena-Hard v2 用 GPT-4.1 + Gemini 2.5 ensemble，不要替换；MixEval-Hard 用 GPT-4-turbo judge；自家自研 judge 上线前跑一遍 Judge Reliability Harness。
3. **code execution sandbox 强制隔离**。LiveBench / LiveCodeBench 的 subprocess 默认不隔离——用 docker / firejail / runuser 至少包一层。
4. **训练数据 ngram filter 是底线但不充分**。13-gram 漏掉 78% 的语义重复（CodeForces 实测），因此 live benchmark + ngram clean 仍然可能拿到 inflated 数字。诚实做法：在 paper / report 中明确写"我们做了 13-gram filter，但未做 semantic dedup"。
5. **没有 dynamic refresh 的 dynamic benchmark 不算 dynamic**。MixEval 自 2024-08 后无新 release，工程上把它当 NeurIPS 2024 时期 baseline，不要拿来声称 anti-contamination。
6. **canary / watermark 是 dataset owner 的事**。Evaluator 角色用 CCV 类黑盒探针更现实，但仍处于 paper 阶段。
7. **承认 judge 不可靠**。LLM-as-judge 在 RAND harness 上的结论是"没有 uniformly reliable judge"——production 用 judge 前必须做 perturbation test，不能假设 SOTA

judge model 在你的 task 上稳。

下一章转入长上下文 / 多模态 / agent 的工程栈。这条路线和本章的"动态题目 + 远端 judge"不同——它的工程问题集中在执行环境（VM / Docker / 浏览器 driver）、KV cache / dimension scaling、多步骤 horizon 上的成本控制。

引用条目

- [^livebench-paper]: White, C., Dooley, S., Roberts, M., Pal, A., Feuer, B., Jain, S., et al. (2025). LiveBench: A Challenging, Contamination-Limited LLM Benchmark. ICLR 2025 Spotlight. https://arxiv.org/abs/2406.19314
- [^livebench-repo]: LiveBench (2024–2026). https://github.com/LiveBench/LiveBench
- [^livecodebench-repo]: LiveCodeBench (2024–2026). https://github.com/LiveCodeBench/LiveCodeBench
- [^mixeval-paper]: Ni, J., Xue, F., Yue, X., et al. (2024). MixEval: Deriving Wisdom of the Crowd from LLM Benchmark Mixtures. NeurIPS 2024. https://arxiv.org/abs/2406.06565
-
- [^arena-hard-paper]: Li, T., Chiang, W.-L., Frick, E., et al. (2024). Arena-Hard and BenchBuilder Pipeline. https://arxiv.org/abs/2406.11939
-
- [^arena-hard-blog]: LMSYS Blog (2024). The Arena-Hard Pipeline. https://www.lmsys.org/blog/2024-04-19-arena-hard/
- [^verbosity-bias]: Saito, K., et al. (2023). Verbosity Bias in Preference Labeling by LLMs. https://arxiv.org/abs/2310.10076
- [^soft-contam-2026]: Spiesberger, A., Vazquez, J. J., Pochinkov, N., et al. (2026). Soft Contamination Means Benchmarks Test Shallow Generalization. https://arxiv.org/abs/2602.12413
- [^ccv-2026]: Song, T.-E. (2026). Cross-Context Verification. https://arxiv.org/abs/2603.21454
- [^closed-watermark-2026]: Huang, P., Chaudhuri, K., Wang, Y.-X. (2026). Dataset Watermarking for Closed LLMs with Provable Detection. https://arxiv.org/abs/2605.06865
- [^llm-olympiad-2026]: Cruz, J. C. B., Aji, A. F. (2026). LLM Olympiad: Why Model Evaluation Needs a Sealed Exam. https://arxiv.org/abs/2603.23292
- [^judge-reliab-2026]: Dev, S., Sloan, A., Kavner, J., Kong, N., Sandler, M. (2026). Judge Reliability Harness. https://arxiv.org/abs/2603.05399; repo: https://github.com/RANDCorporation/judge-reliability-harness
- [^jecs-2026]: Provable Joint Decontamination via Joint Envelope Conformal Selection. https://arxiv.org/abs/2605.21543

53 / 64

第 7 章 · 长上下文 / 多模态 / agent 综合工程

第 6 章的 live / contamination 系列工程问题集中在"题目和 judge"，本章的三大类——长上下文、多模态、agent——工程问题集中在**执行环境**：KV cache 占满 GPU 内存、Docker VM 启停的并发安全、浏览器 driver 的会话状态、长 horizon agent 的 token 成本。本章按这三块分别给代码 / 命令行 / 部署细节，并诚实说明 Stage 0.5 frontier scan 中某些新工作（AgencyBench / AstaBench / MCP-Atlas / Odysseys 等）的执行环境门槛与"无公开 repo"卡片在工程章节的边界。

7.1 NIAH / RULER 的本地跑法与 KV cache 影响

Greg Kamradt 的原版 NIAH (github.com/gkamradt/LLMTest_NeedleInAHaystack) 只是 ~200 行脚本：把 Paul Graham 论文拼成 haystack，在 `(context_length, depth%)` 二维网格扫描，问"What is the most fun thing to do in San Francisco?"。命令行：

```
needlehaystack.run_test \
  --provider openai --model_name gpt-4o-2024-08-06 \
  --document_depth_percents "[0,10,25,50,75,90,100]" \
  --context_lengths_min 1000 --context_lengths_max 128000 \
  --context_lengths_num_intervals 8 \
  --evaluator_name openai
```

KV cache 是这条 pipeline 的关键工程约束。1M-token 长 context 模型 (Claude Opus 1M / Gemini 2.5 / Llama-4 Scout 10M) 跑 NIAH 时，单 prompt 的 KV cache 占用按 70B 模型估算 = `1M tokens × 80 layers × 2 (K+V) × 128 head_dim × 8 num_kv_heads × 2 bytes` ≈ 320 GB——远超单卡 H100 (80GB) 容量。vLLM / SGLang 的 paged attention + KV cache offloading 是必装；自家跑 1M context NIAH 的最小配置：4 × H100 + `--enable-chunked-prefill --max-num-batched-tokens 8192 --gpu-memory-utilization 0.95`。

更狠的是 RULER (github.com/NVIDIA/RULER)，13 任务 × 6 context length = 78 个 cell，每 cell 500 样本。`pred/call_api.py` 是入口，`scripts/run.sh` 套了一层多 GPU 调度。这里的 dimension explosion 工程含义：跑全套 RULER avg 需要 `13 × 6 × 500 = 39,000` queries，按 128K context 大约 5 hours on 8 × H100。如果想报"有效 context size"（首次 average 跌破 85% 的长度），需要至少覆盖 4K → 128K 六档（v2 扩展到 256K / 512K / 1M）；省一档省 ~17% 时间但等于丢掉 degradation curve。

```
git clone https://github.com/NVIDIA/RULER
cd RULER && bash scripts/install.sh
bash scripts/run.sh -m my-model-7b-128k \
  -t niah_single_1,niah_single_2,niah_multikey_1,niah_multivalue,niah_multiquery
  -s 4096,8192,16384,32768,65536,131072 \
  -n 500
```


54 / 64

RULER paper 的关键发现：声称 32K+ 的模型只有约一半在 RULER 32K 维持 ≥85% 平均准确率 `[ruler-paper]` ——比 NIAH 杀伤力大得多。但 2025-2026 的批评论文 (arxiv:2506.02921) 论证 RULER 的合成 needle 与真实任务相关性仅 0.559 `[ruler-critique-2025]`；NoLiMa (arxiv:2502.05167) 进一步指出 NIAH 类只测字面 retrieval、不测语义关联。工程上的应对：在 paper 中除 RULER avg 外，必报 (a) per-task degradation curve；(b) 一个 realistic benchmark (LongBench v2 / NoLiMa)。

LongBench v2 (github.com/THUDM/LongBench) 是 503 道真实长材料 MCQ，跑法简单：

```
python pred.py --model llama3-8b-128k --task longbench_v2 \
  --max_length 131072 --setting cot
```

`--setting cot` 是关键：paper 实测 o1-preview w/ CoT 57.7% > 普通直答最高 50.1% `[longbench-v2-paper]`，证明长 reasoning chain 是长 context 的必要能力。BABILong (github.com/booydar/babilong) 类似，把 bAbI 事实 + PG-19 noise 拼接到任意 token 数；HF dataset 已提供 1K/2K/.../1M/10M 多档预切版本，`load_dataset("RMT-team/babilong", split="qa1/64k")` 即可。

7.2 MMMU 多模态 pipeline 与 test 标签泄漏

MMMU (github.com/MMMU-Benchmark/MMMU) 是 11,550 道大学课程多模态题，2026-02 起公开 test set 标签（之前依赖 EvalAI 远端评测）。一个最小 inference pipeline：

```
from datasets import load_dataset
ds = load_dataset("MMMU/MMMU", "Art", split="test")
# 每条 example 含 question, options (A-D), image_1..image_7 (PIL.Image), answer

def format_mllm_prompt(ex):
    text = ex["question"] + "\n"
    for k, opt in zip("ABCD", [ex["option_A"], ex["option_B"], ex["option_C"], e
        text += f"{k}. {opt}\n"
    text += "Answer with the letter only."
    images = [ex[f"image_{i}"] for i in range(1, 8) if ex.get(f"image_{i}") is n
    return text, images

# 用 vLLM serve 一个 vision model
# vllm serve Qwen/Qwen2-VL-72B-Instruct --tensor-parallel-size 4 --limit-mm-per-
```

MMMU 工程上的两个坑：(a) 部分题最多 7 张图，vLLM `--limit-mm-per-prompt image=7` 必须设，否则会截断；(b) test 标签 2026-02 公开后，**contamination 几乎不可避免** `[mmmu-repo]` ——HF dataset 上 Qwen3.6 Plus 报 86.0% / GPT-5.1 85.4% 已逼近 human expert 88%。工程上应对：和 MMMU 配套必跑 MMMU-Pro (ACL 2025, arxiv:2409.02813)，后者过滤掉 text-only 可答的题目并把选项数从 4 增加到 10，分数从 ~86% 拉回到 16.8-26.9% `[mmmu-pro-paper]`。

```
# MMMU-Pro 是单独 dataset
huggingface-cli download --repo-type dataset MMMU/MMMU_Pro
python evaluate_mmmu_pro.py --model qwen2-vl-72b --setting standard
python evaluate_mmmu_pro.py --model qwen2-vl-72b --setting vision  # vision-only
```

7.3 GAIA 的远端 leaderboard 与 BenchJack exploit

GAIA (huggingface.co/datasets/gaia-benchmark/GAIA) 评测协议是"提交 JSONL → HF Space 远端打分"，validation 165 题答案公开、test 301 题答案私有。最小复现：

```
from huggingface_hub import snapshot_download
from datasets import load_dataset
snapshot_download(repo_id="gaia-benchmark/GAIA", repo_type="dataset", local_dir=
ds = load_dataset("./gaia", "2023_all", split="validation")
# 每题: task_id, Question, Final answer, Level, Annotator Metadata (含 Steps), fi
```

提交格式 `submission.jsonl` 每行 `{"task_id": "...", "model_answer": "...", "reasoning_trace": "..."}`，上传到 https://huggingface.co/spaces/gaia-benchmark/leaderboard。但 2026-04 Berkeley RDI 的 BenchJack paper 揭示：**自动 exploit agent 通过读取 HF 上的 validation answer key 即可达到 ~98% 而完全不解题** `[berkeley-exploit-2026]`。工程上的应对：(a) 不要拿 validation 数字声称 GAIA 能力——必须走 test set；(b) Meta 2026-02 已发布 Gaia2 (arxiv:2602.11964, ICLR 2026 Oral)，引入异步/动态环境 + write-action verifier；GPT-5 (high) overall pass@1 仅 42%，配套开源 Agents Research Environments (ARE) 平台。新模型 benchmark 应迁移到 Gaia2。

7.4 OSWorld 的 VM 启停与并发安全

OSWorld (github.com/xlang-ai/OSWorld) 是 369 个 Ubuntu/Windows/macOS 桌面任务的真实 VM 评测。每 task 两件套：`setup.py`（VM 初始化）+ `evaluator.py`（结束检查）。本地启动：

```
git clone https://github.com/xlang-ai/OSWorld
cd OSWorld && pip install -r requirements.txt

# 下载 Ubuntu VM image (~25 GB)
python desktop_env/providers/vmware/setup.py  # 或 virtualbox / docker / aws

# 跑全套 369 task
python run_multienv.py \
  --provider_name docker \
  --observation_type screenshot_a11y_tree \
  --model claude-sonnet-4 \
  --max_steps 50 --temperature 0
```

OSWorld-Verified (2025-07) 把全套从 10+ 小时压到 ~1 小时，关键是 AWS parallelization。`run_multienv.py` 启动 N 个并发 VM (默认 8)；想拉到 32 并发就需要 32 × (8 GB RAM + 4 vCPU) ≈ 256 GB / 128 vCPU 配置——典型一台 `m5.16xlarge` 单机或 4 × `m5.4xlarge`。

工程上**真正难处理的并发坑**：OSWorld VM 启动时会从 internet 下载特定版本的 LibreOffice / Chrome / VS Code，多个 VM 同时启动会触发 web 反爬（OSWorld-Verified blog 自承 300+ issue 中很多是反爬 `[osworld-verified-2025]`）。建议把所有依赖镜像 cache 到本地 mirror，VM image 内禁用对外 internet 访问，setup script 改为从本地 mirror 拉。

Berkeley RDI 2026-04 的 BenchJack 工作还揭示一个安全坑：**OSWorld 的 agent 在 VM 里跑代码时与 evaluator 共用一个文件系统**——agent 可以读取 evaluator.py 的内容，自动 inject 让 evaluator 总返回 True 的 hack，BenchJack 在 OSWorld 上达 73% 而真实解题率 0 `[berkeley-exploit-2026]`。工程上必须强制 agent VM 与 evaluator 进程隔离（不同 mount namespace），不能简单跑同一份 Python script。

7.5 WebArena / VisualWebArena 自托管 Docker 编排

WebArena (github.com/web-arena-x/webarena) 是 6 个自托管开源站点：Shopping (OneStopShop, Magento 电商)、Shopping Admin (Magento CMS 后台)、Reddit-clone postmill、GitLab、OpenStreetMap、Wikipedia。每个站点是独立 Docker container，启动顺序：

```
# 单机最小版（推荐 64 GB RAM）
docker run -d --name shopping -p 7770:80 shopping_final_0712
docker run -d --name reddit -p 9999:80 webarena-reddit
docker run -d --name gitlab -p 8023:80 webarena-gitlab
docker run -d --name openstreetmap -p 3000:3000 osm_carto-3.0
docker run -d --name wikipedia -p 8888:80 wikipedia_image_dump
# Magento 启动后再跑 'magento setup:upgrade'，约 5 分钟
```

ServiceNow 团队 2026-01 release 的 WebArena-Verified (github.com/ServiceNow/webarena-verified) 把全套打包成 `pip install webarena-verified`，并修复了 506 个 permissive matching bug `[webarena-verified-2026]`。强烈建议**不要用原版**——原版的 evaluator 把"2 000"当"2"接受、92 任务 page-content 字段混淆。

工程上的复现陷阱：WebArena 任务依赖**特定的 DB state**，跑完一个 task 后 DB 被改了，下一个 task 就会失败。`browser_env/auto_login.py` 与 `setup_reset.py` 必须在每个 task 前重置 DB——重置一次 Magento 大约 30 秒。社区报告 (github issues) GitLab/Magento 镜像周期性损坏，2026 之前几乎每个月都有人在 issue 区报"image 拉不下来"。这是个长期维护负担。

VisualWebArena 在此之上加 SoM (Set-of-Marks) representation 给 vision agent，关键代码在 `browser_env/processors.py` 的 `_to_som()` 把 DOM tree 转成带 bounding box + label 的图像。SoM 渲染依赖 `playwright + selenium`，单页面渲染 ~200ms。Berkeley

57 / 64

BenchJack 在 WebArena 上达 ~100% exploit `[berkeley-exploit-2026]`——主要利用答案 key 暴露 + 弱 string match。

7.6 AgentBench 与 τ³-bench 的多 domain 调度

AgentBench (github.com/THUDM/AgentBench) 是 8 个独立环境的 wrapper：OS / DB / KG / Card Game / LTP / ALFWorld / WebShop / Mind2Web。每个环境一个 Docker container + 一份 task config。整体调度走 `src/start_task.py`：

```
# 启所有 8 environment server
docker-compose up -d
# 跑全套
python src/start_task.py -a -d -m gpt-4 -p api
```

2024 后官方维护放缓，**v1 vs FC 数字不可比**——FC（function-calling）分支用 OpenAI function calling schema 重新封装，与原版 chat 模式不可换算。Berkeley BenchJack 2026 工作没把 AgentBench 列入被 break 的 8 个 benchmark `[berkeley-exploit-2026]`——侧面反映其在 2026 影响力已让位于 GAIA / OSWorld / SWE-bench Pro 等专精 benchmark。

τ³-bench (github.com/sierra-research/tau-bench) 2026-03 加 telecom + banking-knowledge + voice 三个新 domain。最小命令行：

```
git clone https://github.com/sierra-research/tau-bench  # 注：旧 repo，新版迁移到 t
pip install -e .
python run.py --agent-strategy tool-calling \
  --env retail --user-strategy llm --user-model claude-3.5-sonnet \
  --agent-model gpt-4o --num-trials 8 \
  --task-ids 0-100
```

τ-bench 评测核心是 **pass^k**：同一 task 跑 k 次（不同 user-simulator seed），所有 k 次都 success 才算 pass^k。工程上要注意：(a) user-simulator 不稳定，多篇 paper (arxiv:2601.17087, 2603.11245) 实测同 task 不同 user-LLM 导致 agent success 波动达 9pp `[lost-in-simulation-2026]`；(b) τ³ 的 voice 子集需要 audio model（OpenAI Realtime API 或 Gemini Live），自托管成本高。

7.7 MCPMark / MCP-Atlas 的 MCP server 一键编排

MCPMark (github.com/eval-sys/mcpmark) 是 127 任务 × 5 MCP server (Notion / GitHub / Filesystem / PostgreSQL / Playwright)：

```
git clone https://github.com/eval-sys/mcpmark
docker compose up -d  # 一键起 5 server + verifier
mcpmark run --task-id all --model gpt-5-medium \
  --pass-at 4 --pass-power 4 --compaction-token 100000
```

关键工程点：repo pin 住 GitHub MCP Server v0.15.0 + Notion @1.9.1 等具体版本号防止评测漂移 `[mcpmark-2026]`。`verify.py` 是程序化验证，agent 跑完后跑 `pytest verify.py::test_*`，二分判定 pass/fail。MCP-Atlas (github.com/scaleapi/mcp-atlas) 规模 10×：1000 task × 36 server / 220 tools，500 hidden 防污染。Docker image `ghcr.io/scaleapi/mcp-atlas:1.2.5`，最低 8 GB Docker memory `[mcp-atlas-2026]`。

两者评估哲学正面冲突：MCP-Atlas 用 claim-based judge 给 partial credit (0/0.5/1.0)，MCPMark 用 programmatic verify 求 0/1。工程上选哪个看 use case——选 frontier model 上限测 MCP-Atlas（hidden split），选 deployment readiness 测 MCPMark（程序化验证更严）。

7.8 长 horizon agent 的成本控制

Stage 0.5 frontier scan 里三个 1M-token / 长 horizon benchmark 需要单独说明成本：

AgencyBench (github.com/GAIR-NLP/AgencyBench, 138 task × 1M context × ~90 tool calls per task × 数小时执行)：跑全套 frontier model（如 Claude Opus 4.7）按 $15/M input + $75/M output 估算单次约 $30+ per task × 138 ≈ **$4,000+ 单 model 评测**。pass@1 单次还要 ×2 (with feedback)。工程上的应对：(a) 用 OpenRouter 跨家比价；(b) 只跑 hard subset (32 scenario 抽样)；(c) 视觉 LLM-judge (Gemini-2.5-Pro) 单独控成本。AgencyBench paper 的 scaffold sensitivity 是另一个工程坑：同一模型在 Claude-Agent-SDK vs 第三方 SDK 上差距可达 20.5% `[agencybench-2026]`，单一 scaffold 跑出的数字不能直接横比。

AstaBench (github.com/allenai/asta-bench, 2400+ 科研 task)：全 suite 需 40-128 GB memory + Docker sandbox + MCP 文献工具 (~4 req/sec rate limit) + 评测大型 log 单次需 30+ min `[astabench-2026]`。Allen AI 自己也承认 E2E discovery 子任务最强 agent 仅 3% 完美完成，但 coding / literature 子任务相对容易，aggregate 看似很高但有"高位虚胖"风险。`astabench score <log_dir>` 自动 aggregate + 算 model usage cost ($/problem)；Claude Opus 4.7 总分 58.0% / $3.54 per problem，GPT-5.5 52.9% / $1.61 per problem。

Odysseys (github.com/ljang0/Odysseys, 200 web task × 100-step budget × ~30min wall-clock)：live Internet 评测，**不可完全 reproducible** `[odysseys-2026]`——网站改版 / 登录墙 / rate-limit 都会让同 task 不同时间结果不同。工程建议：跑前先 freeze 一份 Wayback Machine snapshot 当 baseline；paper claim 要标明跑的 time range。

59 / 64

7.9 没有公开 repo 的 8 张卡——工程章节的边界

`_KNOWN-ISSUES.md` 提示有 8 张卡片没有公开 repo：lemmabench / soohak / eternalmath / riemann-bench（github_url = arxiv URL）、swe-rebench（github_url = swe-rebench.com）、cattle-trade / pac-bench / llm-olympiad（github_url = paper URL）、swe-bench-pro（github_url = blog URL）。工程章节不能假设这些 benchmark 可 clone：

- **swe-rebench** (Nebius) 只有 swe-rebench.com 网站，无 GitHub repo——只能用其公开 leaderboard 报数字；自家集群无法独立复现。
- **swe-bench-pro** (Scale AI) 类似，只挂在 labs.scale.com/leaderboard/swe_bench_pro_public，500 hidden tasks 永远本地拿不到；OpenAI 已停止报告 SWE-bench Verified（confirmed contaminated），Pro 是事实新标准，但工程上只能"提交模型 weights / API 给 Scale 跑分"。
- **lemmabench / soohak / eternalmath / riemann-bench / cattle-trade / pac-bench / llm-olympiad** 都是 paper-only，目前要么作者未释 dataset，要么数据集太小（如 riemann-bench 仅 25 题）专为防污染保私有。

工程章节对这些 benchmark 的态度：只引用 paper 数字，不写命令行——读者如果要跑，得联系作者拿 access。下一章 (Frontier Watch appendix) 会再次诚实声明这些 benchmark 未入 deep tier 的工程原因。

引用条目

- [^niah-repo]: Kamradt, G. (2023). LLMTest_NeedleInAHaystack. https://github.com/gkamradt/LLMTest_NeedleInAHaystack
- [^ruler-paper]: Hsieh, C.-P., Sun, S., et al. (2024). RULER: What's the Real Context Size of Your Long-Context Language Models? COLM 2024. https://arxiv.org/abs/2404.06654
- [^ruler-critique-2025]: A Controllable Examination for Long-Context Language Models (2025). https://arxiv.org/pdf/2506.02921
- [^longbench-v2-paper]: Bai, Y., Tu, S., et al. (2025). LongBench v2. ACL 2025. https://arxiv.org/abs/2412.15204
- [^babilong-paper]: Kuratov, Y., Bulatov, A., et al. (2024). BABILong. NeurIPS 2024 D&B. https://arxiv.org/abs/2406.10149
- [^mmmu-repo]: MMMU-Benchmark/MMMU. https://github.com/MMMU-Benchmark/MMMU
- [^mmmu-pro-paper]: Yue, X., et al. (2025). MMMU-Pro. ACL 2025. https://arxiv.org/abs/2409.02813
- [^gaia2023]: Mialon, G. et al. (2023). GAIA. arXiv:2311.12983
- [^gaia2-2026]: Froger, R. et al. (2026). Gaia2. ICLR 2026 oral. https://arxiv.org/abs/2602.11964
- [^berkeley-exploit-2026]: Wang, H., Mang, Q., Cheung, A., Sen, K., Song, D. (2026). How We Broke Top AI Agent Benchmarks. Berkeley RDI blog.

60 / 64

- [^osworld2024]: Xie, T. et al. (2024). OSWorld. NeurIPS 2024. https://arxiv.org/abs/2404.07972
- [^osworld-verified-2025]: XLANG Lab (2025-07-28). Introducing OSWorld-Verified. https://xlang.ai/blog/osworld-verified
- [^osworld-mcp-2026]: OSWorld-MCP team (2026). ICLR 2026. https://arxiv.org/abs/2510.24563
- [^webarena2023]: Zhou, S., Xu, F. F., Zhu, H. et al. (2023). WebArena. ICLR 2024. https://arxiv.org/abs/2307.13854
- [^webarena-verified-2026]: ServiceNow Research (2026). WebArena Verified. NeurIPS 2025. https://github.com/ServiceNow/webarena-verified
- [^agentbench2023]: Liu, X., et al. (2023). AgentBench. ICLR 2024. https://arxiv.org/abs/2308.03688
- [^tau-bench-2024]: Yao, S., Shinn, N., et al. (2024). τ-bench. https://arxiv.org/abs/2406.12045
- [^tau3-bench-2026]: Sierra Research (2026-03-18). τ^3-Bench. https://sierra.ai/resources/research/tau-3-bench
- [^lost-in-simulation-2026]: Lost in Simulation. arXiv:2601.17087
- [^mcpmark-2026]: Wu, Z., Liu, X., et al. (2026). MCPMark. ICLR 2026. arXiv:2509.24002
- [^mcp-atlas-2026]: Bandi, C. et al. (2026). MCP-Atlas. arXiv:2602.00933
- [^agencybench-2026]: Shi, J., Xiao, Y., et al. (2026). AgencyBench. ACL 2026. arXiv:2601.11044
- [^astabench-2026]: Bragg, J., D'Arcy, M., et al. (2026). AstaBench. ICLR 2026 Oral. arXiv:2510.21652
- [^odysseys-2026]: Jang, L. K., Koh, J. Y., Fried, D., Salakhutdinov, R. (2026). Odysseys. arXiv:2604.24964

附录 · Frontier Watch — 执行环境与工程复杂度

Stage 0.5 frontier scan (2026-02-01 → 2026-05-25) 一共扫到 73 个 2026 新工作 `[frontier-scan]`，本附录从工程角度审：执行环境门槛、依赖体积、并发安全、reproducibility 风险，并诚实声明未入 deep tier 的工程原因。

A. 执行环境门槛分档

| 档位 | 工程门槛 | 例子 | |---|---|---| | 单机 CPU + python | < 1 GB 依赖、可一键跑 | judge-reliability-harness (RAND) | | 单 GPU + Docker | 8–16 GB GPU mem，一台 Docker | mcpmark, livebench | | 多 GPU + 大内存 + VM | 4× H100、64 GB RAM、AWS / VMware |

61 / 64

osworld, ruler (128K), agencybench | | live Internet + 远端 service | 不可完全 reproducible | odysseys, gaia (web tasks), tau3-voice | | API-only / hidden split | 自家集群拿不到数据 | swe-bench-pro, mcp-atlas hidden 500, gaia test |

工程上 deep tier 的隐性门槛：必须能在自家集群独立跑、必须有可复现 evaluator、必须给出可观察的 log。本档的 cut-off 把以下方向挡在 deep tier 之外：

B. 未入 deep tier 的工程原因

B.1 paper-only，无公开 repo。`_KNOWN-ISSUES.md` 列了 8 张：lemmabench / soohak / eternalmath / riemann-bench / swe-rebench / cattle-trade / pac-bench / llm-olympiad / swe-bench-pro。要么数据集太小（riemann-bench n=25）防污染私有，要么 evaluator 只在 paper 描述。工程章节只能引用 paper 数字，不能写命令行。

B.2 完全依赖第三方 service。`swe-bench-pro` 必须把模型提交给 Scale AI 跑分（500 hidden tasks 拿不到）；`gaia` test 答案私有，依赖 HF Space 远端 leaderboard。能用但不能"独立复现"。

B.3 contamination method 在 paper 阶段。CCV (arxiv:2603.21454)、Dataset Watermarking (arxiv:2605.06865)、JECS (arxiv:2605.21543)、Soft Contamination (arxiv:2602.12413) 都声称"release code/data" 但 abstract 页没给 repo URL `[ccv-2026]` `[closed-watermark-2026]`。工程团队当前能落地的还是 lm-evaluation-harness `lm_eval/decontamination/janitor.py` 的 13-gram filter（已知有 78% 漏检 `[soft-contam-2026]`）和 Carlini canary。

B.4 framework 倡议无 reference impl。LLM Olympiad (arxiv:2603.23292) sealed-exam 协议倡议明确说"不提供 reference implementation"；DEP (arxiv:2603.01167) 去中心化协议同样未释 server 端代码。

B.5 live Internet 评测 reproducibility 风险。Odysseys (arxiv:2604.24964) 作者自承"网站改版、登录墙、rate-limit、内容更新都会导致同 task 不同时间结果差异" `[odysseys-2026]`。工程上须 freeze Wayback Machine snapshot 并标明 time range。

C. 依赖体积与并发安全的几个具体数字

- AstaBench: 40–128 GB memory + Docker sandbox + MCP literature tools (~4 req/sec rate limit) `[astabench-2026]`
- OSWorld-Verified: 单 VM 8 GB RAM + 4 vCPU；32 并发 ≈ 256 GB / 128 vCPU；建议 m5.16xlarge 单机 `[osworld-verified-2025]`
- WebArena: 6 Docker container + Magento 启动 ~5 分钟 + 每 task 前 DB reset ~30s
- RULER 128K avg: 8 × H100 跑 ~5 hours，39K queries
- AgencyBench 138 task × 1M context × ~90 tool calls：单 model 全跑 ≈ $4,000+（Claude Opus 4.7 estimate）

62 / 64

- MCP-Atlas: 36 server，20 个零配置 / 11 需 API key / 5 需 key+预载数据，最少 8 GB Docker memory `[mcp-atlas-2026]`

D. 并发安全的工程坑

OSWorld 与 WebArena 共性问题：agent 在 sandbox 内跑代码与 evaluator 共用文件系统，Berkeley BenchJack (2026-04) 利用这个洞达到 OSWorld 73% / WebArena 100% exploit `[berkeley-exploit-2026]`。必须强制 agent VM 与 evaluator 进程隔离（不同 mount namespace）。OSWorld 另一并发坑：多 VM 同时启动会触发 web 反爬（依赖 LibreOffice/Chrome 在线下载），需把依赖镜像 cache 到本地 mirror。

E. 工程结论

2026 工程团队能"今天就上手跑"的 deep tier 集合：livebench, livecodebench, mixeval-hard, arena-hard-v2, niah, ruler, longbench-v2, babilong, mmmu, mmmu-pro, gaia (validation only), osworld-verified, webarena-verified, agentbench, tau-bench v1, mcpmark, mcp-atlas (public 500), astabench, agencybench, odysseys (live), judge-reliability-harness。其余 contamination 新方法 (CCV / JECS / Watermark / Soft Contamination) 值得跟踪，但目前只能引用 paper 结论。

引用条目

- [^frontier-scan]: Stage 0.5 frontier scan, 2026-02-01 → 2026-05-25, 73 new works. (内部 doc: shared/frontier-scan-stage-0_5.md)
- [^ccv-2026]: Song, T.-E. (2026). Cross-Context Verification. https://arxiv.org/abs/2603.21454
- [^closed-watermark-2026]: Huang, P., Chaudhuri, K., Wang, Y.-X. (2026). Dataset Watermarking for Closed LLMs. https://arxiv.org/abs/2605.06865
- [^soft-contam-2026]: Spiesberger, A., et al. (2026). Soft Contamination. https://arxiv.org/abs/2602.12413
- [^odysseys-2026]: Jang, L. K., et al. (2026). Odysseys. https://arxiv.org/abs/2604.24964
- [^astabench-2026]: Bragg, J., et al. (2026). AstaBench. ICLR 2026 Oral. https://arxiv.org/abs/2510.21652
- [^osworld-verified-2025]: XLANG Lab (2025). OSWorld-Verified. https://xlang.ai/blog/osworld-verified
- [^mcp-atlas-2026]: Bandi, C. et al. (2026). MCP-Atlas. https://arxiv.org/abs/2602.00933
- [^berkeley-exploit-2026]: Wang, H., et al. (2026). How We Broke Top AI Agent Benchmarks. Berkeley RDI.

63 / 64

报告截稿日期: 2026-05-25。Frontier Watch 附录覆盖截稿后涌现的新工作。

64 / 64