Plan-STaR：Baseline（只减 demos）之后的实验路线与模块化消融

本文档回答一个非常具体的问题：

> 你已经完成了 **baseline：只减少 demos 数量**（其它设置尽量不改）。接下来应该怎么做实验，才能把“哪些改动真的有效”讲清楚、并且可写进论文？

约束回顾（与你现有实现一致）：

- 训练是**单阶段训练**（不允许 curriculum（课程训练：分阶段接力训））。
- 数据输入固定为 **LeRobotDataset**（不把 Adapter 当贡献点）。
- 不依赖大离线库（不做 retrieval（检索增强：从外部大库捞相似片段））。

本文档会把你的改动拆成可单独开关的模块，并给出：

1. 推荐实验顺序（按时间/成本排序）
2. 每步的具体命令行
3. 每个参数的意义、为什么这么改、可能出现什么结果（含失败模式）

相关更完整的参数/公式细节见：`policy/pi0/docs/plan_star.md` 与 `policy/pi0/docs/fewshot_recipe.md`。

---

0. 环境与命令统一（避免 `uv` 差异）

仓库里有两种常见跑法：`uv run ...` 或直接用虚拟环境 `policy/pi0/.venv`。

为了让命令在你的仓库里**稳定可用**，本文档统一使用：

```
cd policy/pi0
source .venv/bin/activate
python scripts/<xxx>.py ...
```

> 如果你一直用 `bash finetune.sh ...` 没问题，也可以继续；但 `finetune.sh` 不能传入新增的 plan 参数，所以 Plan-STaR 相关实验建议直接用 `python scripts/train.py ...`。

---

1. 你已完成的 baseline 是否合理？

你说的 baseline：**只减少 demos（从 50 抽 5/10/20），其它都不改**，这作为论文/报告的对照是合理的，因为它只改变一个自变量（数据量）。

但注意两点（写论文时要讲清楚）：

1. 你并不是把数据集真的变成 `*-demo_clean-10`，而是通过 plan 在 `*-demo_clean-50` 内做 episode 子集抽样。
   这仍然是“只减 demos”，但**实现路径**是 “plan 选择子集”。

2. 为了保证“其它都不改”，建议把 plan 设置成尽量接近原始 DataLoader 的行为：
    - `stride_S=1`（不跳帧：不额外窗口化）
    - `num_stages_K=1`（权重全相等：不做阶段均衡）
    - `--no-replacement`（无放回：更像原先 shuffle 覆盖）
    - `valid_start_tail_keep=H-1`（尽量不做尾部过滤：更接近原始样本集合）
    - norm stats 先 **reuse（复用）** 50-demo 的统计（避免引入第二个变量）

> 你现在已经按这个思路生成了
> `sampling_plans/beat_block_hammer/onlydemos_demo{D}_es0`，这是一个干净的 baseline。

---

2. Baseline 之后：推荐实验路线（按“先解释掉点 → 再逐个模块提升”）

核心原则：**每次只改一个关键因素**（ablation（消融：逐项开关验证贡献）），否则你很难解释“到底是哪一项带来的提升”。

我建议把路线分成 4 个阶段：

阶段 A：把“步数预算”问题讲清楚（E0：地基实验）

目的：回答“few-shot 掉点主要来自哪里”：是 **steps 太少**、还是 **schedule 不匹配**、还是 **过拟合/遗忘**?

推荐你先固定一个任务（例如 `beat_block_hammer`）做 E0，后续再扩到多任务。

A0.1 50 demos：steps sweep（只改 steps）

```
cd policy/pi0
source .venv/bin/activate

for STEPS in 30000 6000 3000; do
  python scripts/train.py pi0_base_aloha_robotwin_lora \
    --exp-name beat_block_hammer-demo_clean-50_baseline_steps${STEPS}_s42 \
    --overwrite \
    --seed 42 \
    --data.repo-id beat_block_hammer-demo_clean-50 \
    --num-train-steps ${STEPS}
done
```

为什么要做：你后面所有 few-shot 都会跑 6000/3000，如果你不先证明“50 demos 在短 steps 下能达到多少”，审稿人会质疑你选的预算是不是过小或不公平。

A0.2 固定短 steps：schedule 对照（只开/关 budget-aware）

```
cd policy/pi0
source .venv/bin/activate

STEPS=6000
python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-50_steps${STEPS}_sched_off_s42 \
```

```
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS}

python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-50_steps${STEPS}_sched_on_s42 \
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --budget-aware-schedule
```

为什么要做（通俗）：把 30000 步的学习率曲线硬搬到 6000 步，等于“刚热身完就结束/或者衰减不对”，模型容易学不动或乱震荡；budget-aware（预算感知：按总步数重标定）能把这个变量控制住。

A0.3 固定短 steps：容量/冻结对照（只改 freeze-mode）

```
cd policy/pi0
source .venv/bin/activate

STEPS=6000
python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-
50_steps${STEPS}_freeze_default_s42 \
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --budget-aware-schedule --freeze-mode default

python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-
50_steps${STEPS}_freeze_strong_s42 \
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --budget-aware-schedule --freeze-mode strong
```

预期现象：

- `strong` 可能更稳（方差更小），但也可能掉点（欠拟合：学不进去），尤其 clean 上表现为 timeout 增多。

阶段 B：把 few-shot “只减 demos”迁移到短 steps（E1：few-shot 的 compute-limited 基线）

目的：回答“光靠减 demos，在 3000/6000 步会掉成什么样”，为后续模块提升提供参照。

这里你继续用 `onlydemos` plan（不加权、不窗口化），只把 steps 改短：

```
cd policy/pi0
source .venv/bin/activate

for D in 5 10 20; do
  for STEPS in 6000 3000; do
```

```
    python scripts/train.py pi0_base_aloha_robotwin_lora \
      --exp-name beat_block_hammer-
demo_clean-${D}_onlydemos_steps${STEPS}_s42 \
      --overwrite \
      --seed 42 \
      --data.repo-id beat_block_hammer-demo_clean-50 \
      --num-train-steps ${STEPS} \
      --budget-aware-schedule \
      --sampling-plan-path
sampling_plans/beat_block_hammer/onlydemos_demo${D}_es0
  done
done
```

为什么要做（通俗）：你要先证明“短训 + 少数据”到底掉多少，这是后面所有模块改动的基线；否则你无法量化每个模块带来的收益。

阶段 C：模块化提升（严格单因素消融，推荐在 10-shot 上做）

建议固定一个“成本可控但最有代表性”的 setting 来做消融：

- demos：10（10-shot：少样本但不至于完全崩）
- steps：6000（或 3000 更便宜）
- train_seed：42（后续再加 43/44 做方差）
- episode_selection_seed：0（后续再加 1/2 做子集方差）

下面每个模块都给“只改一处”的做法。

模块 1：去相关/覆盖（stride / tail_keep / replacement）

你要回答的问题：少样本短训时，是不是“同一批样本重复太多”（avg_repeat 很大）导致掉点？

对照组设计（只改 plan 生成参数；训练参数不变）：

1. M1-Base：onlydemos（你已有）
2. M1-stride2：只把 stride 从 1 改成 2（更去相关，但 anchors 变少）
3. M1-tail0：只把 tail_keep 从 49 改成 0（更严格过滤尾部）
4. M1-repl：只把 replacement 从 False 改成 True（更“无限流”，但 coverage 更差）

命令示例（以 10 demos 为例）：

```
cd policy/pi0
source .venv/bin/activate

# 1) stride=2
python scripts/build_sampling_plan.py \
  --repo-id beat_block_hammer-demo_clean-50 \
  --train-config-name pi0_base_aloha_robotwin_lora \
  --num-episodes-keep 10 --episode-selection-seed 0 --plan-seed 0 --
sampler-seed 0 \
```

```
  --no-replacement \
  --stride-S 2 \
  --valid-start-tail-keep 49 \
  --stage-strategy time_quantile --num-stages-K 1 \
  --out-dir sampling_plans/beat_block_hammer/m1_stride2_demo10_es0

# 2) tail_keep=0（严格 f<=L-H）
python scripts/build_sampling_plan.py \
  --repo-id beat_block_hammer-demo_clean-50 \
  --train-config-name pi0_base_aloha_robotwin_lora \
  --num-episodes-keep 10 --episode-selection-seed 0 --plan-seed 0 --
sampler-seed 0 \
  --no-replacement \
  --stride-S 1 \
  --valid-start-tail-keep 0 \
  --stage-strategy time_quantile --num-stages-K 1 \
  --out-dir sampling_plans/beat_block_hammer/m1_tail0_demo10_es0

# 3) replacement=True（有放回）
python scripts/build_sampling_plan.py \
  --repo-id beat_block_hammer-demo_clean-50 \
  --train-config-name pi0_base_aloha_robotwin_lora \
  --num-episodes-keep 10 --episode-selection-seed 0 --plan-seed 0 --
sampler-seed 0 \
  --replacement \
  --stride-S 1 \
  --valid-start-tail-keep 49 \
  --stage-strategy time_quantile --num-stages-K 1 \
  --out-dir sampling_plans/beat_block_hammer/m1_repl_demo10_es0
```

训练命令（每个 plan 都一样，只换 `--sampling-plan-path` 与 `--exp-name`）：

```
cd policy/pi0
source .venv/bin/activate

PLAN=sampling_plans/beat_block_hammer/m1_stride2_demo10_es0
python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-10_m1_stride2_steps6000_s42 \
  --overwrite --seed 42 \
  --data.repo-id beat_block_hammer-demo_clean-50 \
  --num-train-steps 6000 \
  --budget-aware-schedule \
  --sampling-plan-path ${PLAN}
```

你需要观察的结果（通俗）：

- 如果 `stride=2` 反而更差：说明关键瞬间被跳过了（漏学）。
- 如果 `tail_keep=0` 更差：说明你原来 anchors 太少（过滤太狠导致重复暴露爆炸）。
- 如果 `replacement=True` 更差：说明你短训需要更强 coverage（覆盖更多不同样本），而不是随机重复抽。

模块 2：阶段/关键片段加权（time_quantile vs change_point_state）

你要回答的问题：少样本时，关键阶段（例如接触/抓取/释放）是不是因为占比太小而学不到？

对照组（只改 plan 的阶段参数）：

- M2-timeK4：`time_quantile` + `K=4`
- M2-cpK4：`change_point_state` + `K=4`
- M2-invfreq：`stage_balance=inv_freq`（更激进补偿稀缺阶段）

命令示例：

```
cd policy/pi0
source .venv/bin/activate

# time_quantile K=4
python scripts/build_sampling_plan.py \
  --repo-id beat_block_hammer-demo_clean-50 \
  --train-config-name pi0_base_aloha_robotwin_lora \
  --num-episodes-keep 10 --episode-selection-seed 0 --plan-seed 0 --
sampler-seed 0 \
  --no-replacement \
  --stride-S 1 --valid-start-tail-keep 49 \
  --stage-strategy time_quantile --num-stages-K 4 --stage-balance
uniform_over_stage \
  --out-dir sampling_plans/beat_block_hammer/m2_timeK4_demo10_es0

# change_point_state K=4
python scripts/build_sampling_plan.py \
  --repo-id beat_block_hammer-demo_clean-50 \
  --train-config-name pi0_base_aloha_robotwin_lora \
  --num-episodes-keep 10 --episode-selection-seed 0 --plan-seed 0 --
sampler-seed 0 \
  --no-replacement \
  --stride-S 1 --valid-start-tail-keep 49 \
  --stage-strategy change_point_state --num-stages-K 4 --change-point-delta
3 --change-point-min-gap 10 \
  --stage-balance uniform_over_stage \
  --out-dir sampling_plans/beat_block_hammer/m2_cpK4_demo10_es0

# inv_freq（更强调稀缺阶段）
python scripts/build_sampling_plan.py \
  --repo-id beat_block_hammer-demo_clean-50 \
  --train-config-name pi0_base_aloha_robotwin_lora \
  --num-episodes-keep 10 --episode-selection-seed 0 --plan-seed 0 --
sampler-seed 0 \
  --no-replacement \
  --stride-S 1 --valid-start-tail-keep 49 \
  --stage-strategy time_quantile --num-stages-K 4 --stage-balance inv_freq
--inv-freq-power 1.0 \
  --out-dir sampling_plans/beat_block_hammer/m2_invfreq_demo10_es0
```

预期现象：

- 如果阶段加权有效，通常表现为 randomized 成功率上升、或 seed 方差下降、或 timeout 降低。
- 如果加权过强（inv_freq 太激进），可能出现“动作抖动/不连贯”（只学转折、不学平稳控制）。

模块 2B（bonus）：夹爪事件加权（gripper_event）

你要回答的问题：任务成败是否主要取决于夹爪开合附近的窗口？如果是，加权是否能显著减少 timeout？

对照组：

- M2B-off：gripper_event 关闭
- M2B-on：开启 + `q=0.95/0.97/0.99`（越大越“挑剔”，越不容易把小抖动当事件）

命令示例：

```
cd policy/pi0
source .venv/bin/activate

for Q in 0.95 0.97 0.99; do
  python scripts/build_sampling_plan.py \
    --repo-id beat_block_hammer-demo_clean-50 \
    --train-config-name pi0_base_aloha_robotwin_lora \
    --num-episodes-keep 10 --episode-selection-seed 0 --plan-seed 0 --
sampler-seed 0 \
    --no-replacement \
    --stride-S 1 --valid-start-tail-keep 49 \
    --stage-strategy time_quantile --num-stages-K 4 --stage-balance
uniform_over_stage \
    --gripper-event-enabled --gripper-dims "6,13" --gripper-quantile-q ${Q}
--gripper-gamma 2.0 \
    --out-dir sampling_plans/beat_block_hammer/m2b_grip_q${Q}_demo10_es0
done
```

预期现象（通俗）：

- 有效：timeout 明显下降（更容易“做到最后一步”），avg_steps_success 也可能下降。
- 无效/变差：可能是 `gripper_dims` 不对或 `q` 太低把小抖动当事件，导致训练分布偏了。

模块 3A/3B：冻结强弱（freeze_mode）

你要回答的问题：少样本是否需要更强正则（regularization：限制模型自由度防止背数据）？

对照组（只改一个参数）：

- M3A：`--freeze-mode default`
- M3B：`--freeze-mode strong`

训练命令示例（保持同一个 plan）：

```
cd policy/pi0
source .venv/bin/activate

PLAN=sampling_plans/beat_block_hammer/m2b_grip_q0.97_demo10_es0
STEPS=6000

python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-
10_freeze_default_steps${STEPS}_s42 \
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --budget-aware-schedule --freeze-mode default \
  --sampling-plan-path ${PLAN}

python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-
10_freeze_strong_steps${STEPS}_s42 \
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --budget-aware-schedule --freeze-mode strong \
  --sampling-plan-path ${PLAN}
```

可能结果：

- `strong` 更稳但更容易欠拟合（clean 掉点，timeout 增）。
- `default` 更容易学进去但 seed 方差可能更大。

模块 4：budget-aware schedule（短训学习率重标定）

你要回答的问题：短训掉点是不是主要来自 schedule 不匹配?

对照组（只开关 `--budget-aware-schedule`）：

```
cd policy/pi0
source .venv/bin/activate

PLAN=sampling_plans/beat_block_hammer/m2b_grip_q0.97_demo10_es0
STEPS=3000

python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-10_sched_off_steps${STEPS}_s42 \
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --sampling-plan-path ${PLAN}

python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-10_sched_on_steps${STEPS}_s42 \
```

```
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --budget-aware-schedule \
  --sampling-plan-path ${PLAN}
```

可能结果：

- 开启后通常更容易收敛（loss 更平稳），success 更高或方差更低。
- 如果变差：可能你的短训本来就需要更大的 peak_lr 或不同 warmup_ratio（作为敏感性再做 0.01/0.03/0.05）。

模块 4B：plan-aware LR scaling（高重复暴露时抑制遗忘）

你要回答的问题：少样本 + plan 采样导致"有效重复次数"很高时，是否需要更小学习率来防止训坏？

对照组（只开关 `--plan-aware-lr-scale`）：

```
cd policy/pi0
source .venv/bin/activate

PLAN=sampling_plans/beat_block_hammer/m1_tail0_demo10_es0   # 举例：anchors
少、重复高时更需要
STEPS=6000

python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-10_lrscale_off_steps${STEPS}_s42
\
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --budget-aware-schedule \
  --sampling-plan-path ${PLAN}

python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-10_lrscale_on_steps${STEPS}_s42 \
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --
num-train-steps ${STEPS} \
  --budget-aware-schedule \
  --plan-aware-lr-scale --plan-target-effective-repeat 50 \
  --sampling-plan-path ${PLAN}
```

可能结果：

- 有效：clean/randomized 掉点缓解（尤其 timeout 降），但学习速度变慢。
- 过小：可能"学不动"（success 低但 loss 也不明显下降）；这时调大 `plan-target-effective-repeat`（50→100）或提高 `plan-lr-min-scale`（0.05→0.1）。

模块 5：norm stats（reuse vs recompute）作为必要控制变量

你要回答的问题：少样本时，归一化统计是稳定器还是噪声源？

对照组：

- reuse：不改 `--data.assets.asset-id`
- recompute：按 plan 子集重算 stats，并在训练里指向新的 asset_id

命令示例（以 10 demos 的某个 plan 为例）：

```
cd policy/pi0
source .venv/bin/activate

PLAN=sampling_plans/beat_block_hammer/m2b_grip_q0.97_demo10_es0

# 1) 重算 norm stats（episodes_valid：按 plan 的 episodes_keep + valid_start 过滤）
python scripts/compute_norm_stats.py pi0_base_aloha_robotwin_lora \
  --plan-path ${PLAN} \
  --plan-subset-mode episodes_valid \
  --output-asset-id beat_block_hammer-demo_clean-10_m2b_stats

# 2) 训练时指向新 stats
python scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-10_norm_recompute_steps6000_s42 \
  --overwrite --seed 42 --data.repo-id beat_block_hammer-demo_clean-50 --num-train-steps 6000 \
  --budget-aware-schedule \
  --data.assets.asset-id beat_block_hammer-demo_clean-10_m2b_stats \
  --sampling-plan-path ${PLAN}
```

可能结果：

- recompute 变好：说明子集分布和全量差异大（例如 state/action 范围差异），子集统计更贴合。
- recompute 变差：说明少样本统计噪声大（std/分位数不稳定），复用全量统计反而更稳。

阶段 D：把消融扩展成论文级主矩阵（5/10/20 + 多 seed）

当你在 10-shot 上找到“最有效的 1~2 个模块组合”之后，再扩展到主矩阵：

- demos：5 / 10 / 20 / 50
- steps：6000 与 3000（或加 30000 作为上限参照）
- seeds：至少 3 个 train_seed（42/43/44）
- episode_selection_seed：至少 3 个（0/1/2）
- 评测：clean + randomized

> 这一步请务必把 run_summary.json（暴露量指标）与 `_metrics.json`（timeout/耗时）一起进表，否则很难解释为什么某些组 success 低是“超时多”还是“异常多”。

---

3. 评测与记录：每组训练后都做这两件事

/

3.1 评测 clean + randomized，并记录失败类型

```
cd policy/pi0

# clean
bash eval.sh beat_block_hammer demo_clean pi0_base_aloha_robotwin_lora \
  <EXP_NAME> 0 0 -1

# randomized
bash eval.sh beat_block_hammer demo_randomized pi0_base_aloha_robotwin_lora
\
  <EXP_NAME> 0 0 -1
```

你要看的不是只有成功率，还要看：

- `_metrics.json` 里的 `failure_counts.timeout`（超时是否减少）
- `avg_steps_success`（成功是否更快）
- `eval_time_sec`（是否推理变慢）

3.2 把 run_summary 与 plan_meta 里的“暴露量/anchors”写进总表

推荐最小字段（写论文表格足够）：

- data：`task_name, repo_id, demos(D), episode_selection_seed`
- budget：`num_train_steps, batch_size`
- plan：`plan_sha256, plan_num_anchors, exposure_avg_repeat, exposure_ESS`
- method toggles：`stride_S, tail_keep, replacement, stage_strategy, K, stage_balance, gripper_event, freeze_mode, budget_aware_schedule, plan_aware_lr_scale, norm_stats(reuse/recompute)`
- eval：`succ_clean, succ_rand, timeout_clean, timeout_rand, avg_steps_clean, avg_steps_rand, eval_time`

---

4. 参数速查表（你“改了什么”以及为什么）

下面只列你后续实验里最常改的参数（完整解释见 `plan_star.md`）。

4.1 plan 生成（`scripts/build_sampling_plan.py`）

- `--num-episodes-keep`：保留 demos 数（5/10/20）。
  影响：少样本越小，方差越大、越容易过拟合。
- `--episode-selection-seed`：决定“抽哪几条 demo”。
  影响：少样本下子集差异很大，必须多 seed 才能得出结论。
- `--stride-S`：隔几帧取一个 anchor（锚点：窗口起点）。
  预期：stride 大更去相关（信息密度高），但 anchors 少会让重复暴露变大、并可能漏关键瞬间。
- `--valid-start-tail-keep`：放宽尾部起点。
  预期：增大 anchors、降低重复暴露；但太大可能引入“尾部无意义动作”比例，需要看 timeout/成功率。

- `--replacement/--no-replacement`：是否有放回抽样。
  预期：replacement=True 更像无限流；replacement=False 更利于 coverage（覆盖更多不同样本）。
- `--stage-strategy` + `--num-stages-K` + `--stage-balance`：阶段定义与阶段均衡。
  预期：更强调关键阶段可减少“关键动作学不到”；过强会导致控制不平滑。
- `--gripper-event-enabled` + `--gripper-quantile-q` + `--gripper-gamma`：夹爪事件加权。
  预期：减少 timeout（更容易完成最后一步）；q 太低会把小抖动当事件导致偏置。

4.2 训练（`scripts/train.py`）

- `--num-train-steps`：训练步数（你的目标预算是 3000/6000）。
  预期：越小越快，但更依赖合理采样与 schedule。
- `--budget-aware-schedule`：按总步数重标定 warmup/decay。
  预期：短训更稳、更容易收敛。
- `--freeze-mode default/strong`：冻结强弱。
  预期：strong 更正则、更稳但可能欠拟合；default 上限更高但更容易过拟合。
- `--plan-aware-lr-scale`：根据 plan 的重复暴露（ESS）自动缩小学习率。
  预期：减少遗忘/过拟合；太保守会学不动。
- `--data.assets.asset-id`：切换 norm stats（reuse vs recompute）。
  预期：recompute 可能更贴合子集，但也可能因样本少而更噪。