Plan-STaR（Plan-driven Stage-aware Reweighting）使用说明

本说明文档对应仓库内的“采样计划文件（sampling plan）驱动的单阶段训练”实现：训练流程本身不做课程训练（curriculum：分阶段接力训练），只通过**离线生成的 plan 文件**固定“训练时看哪些样本、各样本权重、阶段标签与随机种子”，从而在 few-shot（少量示范）+ short-run（更少训练步数）预算下提升有效样本覆盖与可复现性。

> 重要：**不传 `sampling_plan_path` 时，训练行为与原 baseline 完全一致**（仍按原来的 `shuffle=True` 读取数据）。

---

0. 实现核对（对照你的要求）

下面按“你的硬约束/关注点”逐条核对当前实现（并给出对应代码位置，方便你审阅）：

- 不干扰 baseline：`sampling_plan_path` 默认是 `None`，训练只有在你显式传入 `--sampling-plan-path` 时才启用 plan 采样；否则仍然走原本的 `shuffle=True`。
  - 入口参数：`policy/pi0/src/openpi/training/config.py`（`TrainConfig.sampling_plan_path` 默认 `None`）
  - 采样开关：`policy/pi0/src/openpi/training/data_loader.py`（`if config.sampling_plan_path: ... else: shuffle`）
- 单阶段训练（无 Curriculum）：训练脚本不做 Phase0/1/2 或多阶段接力训练；plan 只影响“数据出现的分布”（采样器），训练循环仍是一段 `for step in range(num_train_steps)`。
  - 训练循环：`policy/pi0/scripts/train.py`
  - plan 只在 DataLoader/Sampler 生效：`policy/pi0/src/openpi/training/data_loader.py`
- 输入只依赖 LeRobotDataset（无 Adapter 作为贡献点）：plan 生成与训练读取均直接使用 `lerobot.common.datasets.lerobot_dataset.LeRobotDataset`。
  - plan 生成：`policy/pi0/scripts/build_sampling_plan.py`
  - 训练读数：`policy/pi0/src/openpi/training/data_loader.py`
- 不依赖大离线库 / 无检索增强（retrieval）：plan 只从当前 repo_id 的数据集中抽取 episode/frame/state 来构建索引与权重，不会访问额外大数据池。
  - plan 生成只读 `hf_dataset` 列：`policy/pi0/scripts/build_sampling_plan.py`
- Plan 文件包含“50→5/10/20 demo 子集抽样”的逻辑：通过 `--num-episodes-keep` + `--episode-selection-seed` 从 50 条 episode 中抽子集，并写入 `plan_meta.json` 的 `episodes_keep`。
  - episode 抽样与落盘：`policy/pi0/scripts/build_sampling_plan.py`
- 夹爪事件加权：plan 生成支持 `--gripper-event-enabled`，对“窗口内包含夹爪开合关键变化”的样本提高权重，并通过中值滤波 + 分位数阈值 + NMS 抑制“正常运动时的小抖动”被误判为关键事件。
  - 事件检测与加权：`policy/pi0/scripts/build_sampling_plan.py`
- WeightedRandomSampler 的 replacement 与可复现：plan 里记录 `replacement/sampler_seed`；训练时构造 `WeightedRandomSampler(..., replacement=..., generator=seeded)`，并在 DataLoader worker 中同步设置 `random/numpy/torch` 种子，减少多进程随机性。
  - sampler：`policy/pi0/src/openpi/training/data_loader.py`

    - worker seed：policy/pi0/src/openpi/training/data_loader.py
- norm stats（归一化统计）进入控制变量：新增 compute_norm_stats.py 支持按 plan 子集重算（anchors/episodes/episodes_valid），训练时可用 --data.assets.asset-id 指向重算结果，实现 reuse vs recompute 的严格对照。
    - 重算脚本：policy/pi0/scripts/compute_norm_stats.py
    - 训练加载位置：policy/pi0/src/openpi/training/config.py（DataConfigFactory._load_norm_stats）
- 评测侧补充 “耗时/avg_steps/失败类型” 并支持指定 ckpt：eval.sh 允许第 7 个参数覆盖 checkpoint_id；checkpoint_id=-1/latest 会自动选最大步数 ckpt；评测会额外落盘 _metrics.json，包含 eval_time_sec / avg_steps_success / failure_counts。
    - eval.sh：policy/pi0/eval.sh
    - latest 选择：policy/pi0/pi_model.py
    - metrics：script/eval_policy.py

> 你可以用 git diff 或直接打开以上文件，逐段核查“是否只改采样分布、是否保持单阶段训练、是否默认不启用 plan”。

---

1. 代码入口与新增文件

- 采样计划数据结构：policy/pi0/src/openpi/training/sampling_plan.py
- 生成 plan：policy/pi0/scripts/build_sampling_plan.py
- 校验 plan：policy/pi0/scripts/verify_sampling_plan.py
- 校验 LeRobotDataset 字段：policy/pi0/scripts/verify_lerobot_episode_fields.py
- 训练接入（plan-driven sampler）：policy/pi0/src/openpi/training/data_loader.py
- 训练侧记录 run_summary 与暴露量指标：policy/pi0/scripts/train.py
- 统计 norm stats（支持按 plan 子集）：policy/pi0/scripts/compute_norm_stats.py
- 评测支持 checkpoint_id=-1/latest 与指标落盘：policy/pi0/eval.sh、policy/pi0/pi_model.py、script/eval_policy.py

---

2. 先做一次数据字段自检（推荐）

Plan-STaR 的 episode 子集抽样与阶段标注依赖 episode_index / frame_index 等字段（LeRobotDataset 通常具备）。

在仓库根目录执行（会使用 policy/pi0/.venv 的依赖）：

```
source policy/pi0/.venv/bin/activate
python policy/pi0/scripts/verify_lerobot_episode_fields.py --repo-id
beat_block_hammer-demo_clean-50
```

如果你想直观看到“action_horizon 跨越 episode 末尾会退化”的现象（用于解释为何需要 f <= L_e - H 的有效起点过滤）：

```
python policy/pi0/scripts/verify_action_horizon_boundary.py --repo-id
beat_block_hammer-demo_clean-50 --train-config-name
pi0_base_aloha_robotwin_lora
```

如果你的数据集**缺少** `episode_index/frame_index`，`build_sampling_plan.py` 会退化为"单 episode 模式"（无法做 50→5/10/20 的 demo 子集抽样），此时必须使用 `--num-episodes-keep 1`。

---

3. 生成 Sampling Plan（包含 5/10/20 demo 子集 + 权重/阶段/种子）

3.1 从 `*-demo_clean-50` 抽取 10 条 demo（episode 子集抽样）

```
cd policy/pi0
uv run 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 \
  --stride-S 2 \
  --stage-strategy time_quantile \
  --num-stages-K 4 \
  --stage-balance uniform_over_stage \
  --out-dir sampling_plans/beat_block_hammer/demo10_s0
```

生成结果目录包含：

- `sampling_plans/beat_block_hammer/demo10_s0/plan_meta.json`
- `sampling_plans/beat_block_hammer/demo10_s0/plan_arrays.npz`

校验：

```
uv run scripts/verify_sampling_plan.py --plan-path
sampling_plans/beat_block_hammer/demo10_s0
```

3.2 使用 change-point（变化点：状态变化突增）做阶段边界

change-point（变化点：用"状态变化幅度突然变大"来找关键时刻）更贴近抓取/释放等关键瞬间，但可能对噪声敏感。

```
cd policy/pi0
uv run 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 \
  --stride-S 2 \
  --stage-strategy change_point_state \
  --num-stages-K 4 \
  --change-point-delta 3 \
  --change-point-min-gap 10 \
  --out-dir sampling_plans/beat_block_hammer/demo10_s0_cp
```

3.3 开启“夹爪事件加权”（gripper event weighting）

gripper event（夹爪事件：夹爪开合等关键变化时刻）会对“窗口内包含关键事件”的样本提高权重；实现中包含：

- median smoothing（中值滤波：抑制小抖动）
- multi-scale diff（多尺度差分：用不同时间间隔看变化）
- quantile threshold（分位数阈值：只保留最显著的一小部分变化）
- NMS（非极大值抑制：避免同一事件被密集重复计数）

示例（`gripper_dims` 需要按你的 `observation.state` 维度定义）：

```
cd policy/pi0
uv run 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 \
  --stride-S 2 \
  --stage-strategy time_quantile \
  --gripper-event-enabled \
  --gripper-dims "6,13" \
  --gripper-quantile-q 0.95 \
  --gripper-min-gap 10 \
  --gripper-event-mode soft_exp \
  --gripper-sigma 10 \
  --gripper-gamma 2.0 \
  --out-dir sampling_plans/beat_block_hammer/demo10_s0_grip
```

---

4. Norm stats（归一化统计）控制变量：reuse vs recompute

norm stats（归一化统计：用于把 state/action 做标准化的均值/方差等）会显著影响 few-shot 的稳定性，因此建议作为必要对照：

- reuse（复用：直接用已有的 norm_stats，不重新计算）
- recompute（重算：按当前数据/子集重新计算并落盘）

4.1 全量重算（覆盖整个 repo_id）

```
cd policy/pi0
uv run scripts/compute_norm_stats.py pi0_base_aloha_robotwin_lora
```

4.2 按 plan 子集重算（推荐用 episodes_valid）

episodes_valid 会额外过滤掉 episode 末尾不足以提供 action_horizon 的起点（valid_start 规则）。默认严格规则是 f <= L_e - H；如果你的 plan 在 plan_meta.json 里记录了 valid_start_rule.tail_keep，则会按 f <= min(L_e - H + tail_keep, L_e - 1) 放宽尾部起点（用来增大 anchors、降低重复暴露）。

```
cd policy/pi0
uv run scripts/compute_norm_stats.py pi0_base_aloha_robotwin_lora \
  --plan-path sampling_plans/beat_block_hammer/demo10_s0 \
  --plan-subset-mode episodes_valid \
  --output-asset-id beat_block_hammer-demo_clean-10_plan10_s0
```

> 说明：output_asset_id 用来避免覆盖原来的 norm_stats.json。训练时需要把 asset_id 指向这个新目录，例如追加 --data.assets.asset-id beat_block_hammer-demo_clean-10_plan10_s0。

---

5. 训练：baseline 不变 + plan 驱动训练

5.1 原 baseline（不使用 plan）

原流程不变：

```
cd policy/pi0
bash finetune.sh pi0_base_aloha_robotwin_lora beat_block_hammer-demo_clean-50_baseline 0
```

5.2 使用 plan（few-shot + 短步数）

直接用 uv run scripts/train.py ... 传入新增参数即可：

```
cd policy/pi0
XLA_PYTHON_CLIENT_MEM_FRACTION=0.45 uv run scripts/train.py pi0_base_aloha_robotwin_lora \
  --exp-name beat_block_hammer-demo_clean-10_planstar_s0_steps6000 \
  --overwrite \
  --num-train-steps 6000 \
  --budget-aware-schedule \
  --freeze-mode default \
```

```
  --plan-aware-lr-scale \
  --sampling-plan-path sampling_plans/beat_block_hammer/demo10_s0
```

> 说明：`freeze-mode strong` 会额外冻结视觉塔（vision tower：图像编码器），速度更快但可能掉点；建议作为消融项（3A/3B）而不是默认。

训练目录会额外写入：

- `policy/pi0/checkpoints/<train_config>/<exp_name>/run_summary.json`（包含 plan_sha256 与暴露量指标：ESS/expected_unique/expected_coverage/avg_repeat）
- `policy/pi0/checkpoints/<train_config>/<exp_name>/sampling_plan/`（尝试复制 plan 文件，便于复现审计）

---

6. 评测：支持指定 checkpoint 或 latest

`policy/pi0/deploy_policy.yml` 默认 `checkpoint_id: 3000`。现在你可以在 `eval.sh` 的第 7 个参数覆盖它：

```
cd policy/pi0
bash eval.sh beat_block_hammer beat_block_hammer_clean
pi0_base_aloha_robotwin_lora \
  beat_block_hammer-demo_clean-10_planstar_s0_steps6000 0 0 6000
```

评测 latest（`checkpoint_id=-1` 或 `latest`）：

```
cd policy/pi0
bash eval.sh beat_block_hammer beat_block_hammer_clean
pi0_base_aloha_robotwin_lora \
  beat_block_hammer-demo_clean-10_planstar_s0_steps6000 0 0 -1
```

评测输出目录（示例）：

- `_result.txt`：原来的成功率输出仍保留
- `_metrics.json`：新增落盘指标（`eval_time_sec`、`failure_counts{timeout/exception/other}`、`avg_steps_success` 等）

---

7. 常见报错与排查

- `Sampling plan repo_id mismatch`：plan 的 `repo_id` 与训练 config 的 `data.repo_id` 不一致；重新生成 plan 或改训练配置。
- `Sampling plan dataset_fingerprint mismatch`：数据版本变了（缓存更新/重新生成数据）；重新生成 plan。
- `Not enough anchors for one batch`：少样本 + stride 太大导致锚点不足；降低 `stride_S` 或增加 `num_episodes_keep`。

- `Normalization stats not found`：先运行 `scripts/compute_norm_stats.py` 生成 `norm_stats.json`。

---

8. 参数说明（专业版）

本节的目标是：让你能把每个参数写进论文的“实现细节 / 复现设置（reproducibility：可复现性设置）”里，并清楚它到底改变了什么。

8.1 `scripts/build_sampling_plan.py`（plan 生成脚本）

你可以先运行 `uv run scripts/build_sampling_plan.py --help` 查看完整列表。本实现里参数可以按 5 组理解：

(A) 输入与 few-shot 子集（episode 子集抽样）

- `--repo-id`：LeRobotDataset 的 `repo_id`（例如 `beat_block_hammer-demo_clean-50`）。
- `--train-config-name`：训练配置名（例如 `pi0_base_aloha_robotwin_lora`），用于读取：
  - `action_horizon H`（窗口长度，来自 `config.model.action_horizon`）
  - `batch_size`（用于默认 `min_anchors=batch_size`）
- `--num-episodes-keep`：保留多少条 demo（episode）。典型取值：`5/10/20/50`。
- `--episode-selection-seed`：episode 子集抽样的随机种子（同一数据与种子下，`episodes_keep` 固定）。

实现细节（公式）：

```
episodes_all = unique(episode_index)
episodes_keep = choice(episodes_all, size=num_episodes_keep, replace=False,
seed=episode_selection_seed)
```

(B) 有效起点过滤（避免 horizon 越界导致标签退化）

对每个 episode e 的长度记为 L_e，窗口长度为 H（= action_horizon）。只允许从满足下式的 frame 作为训练样本起点：

```
tail_keep = clamp(valid_start_tail_keep, 0, H-1)
valid_start(e, f) := (f <= min(L_e - H + tail_keep, L_e - 1))
```

对应参数：

- `--valid-start-tail-keep`：在严格规则 `f <= L_e - H` 的基础上，额外允许保留多少帧作为起点（`0` 表示严格过滤）。
  - 对于 `H=50`，设置为 `49` 近似等价于“不做尾部过滤”（允许起点到 `L_e-1`）。
  - 推荐先做敏感性：`0`（严格） vs `H-1`（放宽）。

这条规则会写入 `plan_meta.json` 的 `valid_start_rule.formula`，并直接影响 `anchor_indices` 的候选集合。

(C) 去相关锚点（stride/anchor windowing）

- `--stride-S`：episode 内锚点间隔（stride：隔几帧取一个样本）。值越大，相邻样本越不相关，但 anchors 数更少。
- `--plan-seed`：影响每个 episode 的 stride 对齐偏移（offset），用于“去相关且可复现”。
- `--min-anchors`：锚点最少要有多少个（默认是 `batch_size`，保证至少能凑够 1 个 batch）。
  - 若 anchors 不足，脚本会自动把 stride 减半直到够用或降到 1（避免因为 stride 太大导致无法训练）。

实现细节（公式）：

```
offset_e = hash(plan_seed, episode_id=e) mod S
anchor(e, f) := valid_start(e, f) AND ((f - offset_e) mod S == 0)
```

(D) 阶段标注（stage labeling）与阶段均衡（stage-balanced weights）

- `--stage-strategy`：阶段定义方式
  - `time_quantile`：时间分位数（把每条轨迹按时间均分为 K 段；跨任务最稳）。
  - `change_point_state`：变化点（用 `observation.state` 的变化幅度突增检测边界；更语义化但更敏感）。
- `--num-stages-K`：阶段数 K（推荐 `4` 或 `6` 做消融）。
- `--change-point-delta`（仅 `change_point_state`）：差分步长 Δ（例如 3）。
- `--change-point-min-gap`（仅 `change_point_state`）：相邻边界最小间隔（避免边界密集抖动）。

时间分位数（公式）：

```
denom_e = (L_e - H)
stage(e, f) = clip( floor(K * f / denom_e), 0, K-1 )
```

变化点（概要公式，实际还包含峰值筛选 + NMS）：

```
score[t] = || state[t] - state[t-Δ] ||_2
boundaries = top-(K-1) peaks(score) with NMS(min_gap)
stage(e, f) = number of boundaries <= f
```

阶段均衡（权重基底）：

- `--stage-balance`
  - `uniform_over_stage`：让每个阶段在总体采样概率上尽量均匀。
  - `inv_freq`：按频率逆比（可更激进补偿稀缺阶段）。

- `--inv-freq-power`：`inv_freq` 的指数 p（p 越大补偿越强）。

实现（公式）：

```
count_k = number of anchors with stage_id = k
w_base(i) =
  1 / count_{stage(i)}                        (uniform_over_stage)
  1 / (count_{stage(i)} ^ inv_freq_power)     (inv_freq)
```

数值稳定（避免极端权重）：

- `--clip-ratio`：把权重裁剪到 `[median/clip_ratio, median*clip_ratio]`。
- `--epsilon-mix`：把权重与均值做 ε 混合，避免某些样本权重过小或为 0。

(E) 夹爪事件加权（gripper event weighting）

- `--gripper-event-enabled`：是否启用夹爪事件加权。
- `--gripper-dims`：夹爪在 `observation.state` 中的维度索引（例如 `"6,13"`，需按数据集确认）。
  - 建议先用 `verify_lerobot_episode_fields.py` 看 `state_dim`，再人工确认哪几维对应夹爪。
- `--gripper-smooth-window`：中值滤波窗口（抑制小抖动）。
- `--gripper-delta1 / --gripper-delta2`：两种时间间隔的差分（multi-scale diff：多尺度差分）。
- `--gripper-quantile-q`：分位数阈值 q（只保留最显著变化；q 越大越“挑剔”）。
- `--gripper-min-gap`：事件峰值的最小间隔（NMS：非极大值抑制，避免一个事件被密集重复计数）。
- `--gripper-event-mode`
  - `soft_exp`：距离越近权重越大（平滑衰减）。
  - `hard`：窗口距离事件 ≤ w 才加权（硬阈值）。
- `--gripper-sigma`（soft）：衰减尺度 σ。
- `--gripper-window-w`（hard）：阈值窗口 w。
- `--gripper-gamma`：事件加权强度（最终倍数为 `1 + gamma * m`）。

实现（公式）：

```
g[t] = max_d | state[t, d] |, d in gripper_dims
g_smooth = median_filter(g, window=gripper_smooth_window)
d[t] = max( |g_smooth[t]-g_smooth[t-Δ1]|, |g_smooth[t]-g_smooth[t-Δ2]| )
tau = quantile( d[Δmax:], q )
events = peaks(d > tau) after NMS(min_gap)

dist(i) = min distance from event to window [f_i, f_i+H-1]
m(i) =
  exp( -dist(i) / sigma )      (soft_exp)
  1{ dist(i) <= w }            (hard)
w_event(i) = 1 + gamma * m(i)
```

最终权重：

```
w(i) = clip( w_base(i) * w_event(i), median/clip_ratio, median*clip_ratio )
w(i) = (1-ε)*w(i) + ε*mean(w)     (ε = epsilon_mix)
```

(F) 采样器与复现相关参数

- `--sampler-seed`：plan 里记录的 sampler 随机种子（训练时用于 `WeightedRandomSampler(generator=...)`）。
- `--replacement / --no-replacement`：是否“有放回采样”（replacement：抽完一个样本后它还能再被抽到）。
  - few-shot 短训通常建议 `replacement=True`（更像“无限流”采样）。
  - `replacement=False` 更像“每轮 epoch 看一遍全量 anchors”，更利于 coverage，但在很短 steps 下可能不够灵活。
- `--out-dir`：plan 输出目录，会生成 `plan_meta.json` + `plan_arrays.npz`。

plan 文件本身也会记录 `plan_sha256`（哈希），用于审计“这次实验到底用了哪个采样协议”。

8.2 `scripts/train.py`（训练脚本 / Tyro 参数）

训练入口示例：

```
uv run scripts/train.py pi0_base_aloha_robotwin_lora --exp-name <EXP_NAME>
[其它参数...]
```

你可以用 `uv run scripts/train.py <train_config_name> --help` 查看全部参数。下面解释对 Plan-STaR 复现实验最关键的参数（其它如 optimizer/lr_schedule 的字段含义与原项目一致）。

(A) 实验 I/O 与训练预算

- 位置参数 `<train_config_name>`：选择 `TrainConfig`（例如 `pi0_base_aloha_robotwin_lora`，定义在 `policy/pi0/src/openpi/training/config.py`）。
- `--exp-name`：实验名（决定 checkpoint 目录：`policy/pi0/checkpoints/<train_config>/<exp_name>/...`）。
- `--overwrite`：覆盖同名 checkpoint 目录。
- `--resume`：从最后一个 checkpoint 继续训练。
- `--num-train-steps`：训练步数（你 baseline 是 30000；short-run 常用 6000/3000）。
- `--batch-size`：全局 batch size（必须能被 device_count 整除）。
- `--num-workers`：DataLoader workers 数。
- `--seed`：训练随机种子（用于 JAX RNG、DataLoader base seed 等）。

(B) Plan-STaR：plan 驱动采样（单阶段）

- `--sampling-plan-path`：plan 路径（目录或 `plan_meta.json` 或 `plan_arrays.npz`）。
  - 启用后训练会：
    1. 校验 plan 与当前训练数据匹配（repo_id / dataset_len / action_horizon / fingerprint）
    2. `Subset(dataset, anchor_indices)`
    3. `WeightedRandomSampler(weights, replacement=..., generator=seeded)`
  - 对应实现：`policy/pi0/src/openpi/training/data_loader.py`
- `--sampler-seed-override`：覆盖 plan 中的 `sampler_seed`（用于控制 sampler 抽样随机性）。
- `--sampler-replacement-override`：覆盖 plan 中的 `replacement`（是否有放回抽样）。

训练时会写入：

- `run_summary.json`：包含 `plan_sha256` 与“有效样本暴露量”指标（ESS/expected_unique/expected_coverage/avg_repeat）。
  - 指标定义：

```
n = num_train_steps * batch_size
p_i = w_i / sum_j w_j
ESS = 1 / sum_i p_i^2
E[unique] =
  sum_i (1 - (1 - p_i)^n)      (replacement=True)
  min(N, n)                    (replacement=False, 本实现 num_samples=N)
coverage = E[unique] / N
avg_repeat = n / E[unique]
```

(C) Freeze（冻结）强弱：3A/3B（保证落地）

- `--freeze-mode default|strong`
  - `default`：保持 config 里原本的 `freeze_filter`（baseline 不变）。
  - `strong`：在原 `freeze_filter` 基础上额外冻结视觉塔（vision tower：图像编码器），减少可训练参数、降低少样本过拟合风险。
  - 对应实现：`policy/pi0/scripts/train.py`（通过 path regex `.*PaliGemma/img.*` 额外冻结）。

(D) budget-aware schedule（短训学习率配方重标定）

- `--budget-aware-schedule`：开启后，会根据你的 `--num-train-steps` 自动重标定 warmup/decay，使 30000→6000/3000 时不会因为 warmup/decay 不匹配导致“不收敛/震荡”。
- `--budget-warmup-ratio`：warmup 占总步数比例（默认 0.03）。
- `--budget-min-warmup-steps`：warmup 最小步数（默认 50）。

对应实现：`policy/pi0/scripts/train.py`，逻辑：

```
warmup_steps = clamp( max(min_warmup, round(warmup_ratio * total_steps)),
1, total_steps/2 )
CosineDecay:  decay_steps = total_steps
RsqrtDecay:   timescale = total_steps
```

(D2) plan-aware LR scaling（plan 感知学习率缩放，可选）

当启用 Sampling Plan 时，常见现象是 anchors 数很少，但训练步数仍然很大，导致同一批样本被反复看到（有效重复次数很高），从而出现：

- 过拟合（overfitting：对训练 demo 记住了但泛化差）
- 遗忘（catastrophic forgetting：把 base 模型原本能力训没了）

本实现提供一个**默认关闭**的可消融开关（不影响 baseline）：

- `--plan-aware-lr-scale`：开启后会根据 plan 的 ESS 估算有效重复次数 `repeat_eff = n / ESS`，并缩放学习率：

```
n = num_train_steps * batch_size
ESS = 1 / sum_i p_i^2,  p_i = w_i / sum_j w_j
repeat_eff = n / ESS
scale = clip( (target_repeat / repeat_eff) ^ power, lr_min_scale, 1 )
peak_lr <- peak_lr * scale
decay_lr <- decay_lr * scale
```

对应参数：

- `--plan-target-effective-repeat`：target_repeat（建议 30~100 做敏感性）
- `--plan-lr-scale-power`：power（默认 0.5，开平方缩放）
- `--plan-lr-min-scale`：lr_min_scale（默认 0.05）

> 建议先在 10-demo/20-demo + 6000 steps 上启用它，看看 clean/randomized 掉点是否明显缓解。

(E) norm stats（归一化统计）reuse vs recompute

- `--data.assets.asset-id`：训练时从 `policy/pi0/assets/<train_config_name>/<asset_id>/norm_stats.json` 加载归一化统计。
  - reuse：保持默认 asset_id（通常等于 repo_id）。
  - recompute：运行 `compute_norm_stats.py --output-asset-id <new_id>` 后，在训练时传 `--data.assets.asset-id <new_id>`。

> 注意：`--data.repo-id` 也可以切换训练数据 repo；但请保证与 plan 的 `repo_id` 一致，否则会触发 mismatch。

---

9. 参数说明（通俗版）

这一节把上面的“公式/细节”换成更容易上手的解释：你只要知道“哪些旋钮会让模型更快学到关键动作”，就能做 few-shot 的矩阵与消融。

9.1 生成 plan（`build_sampling_plan.py`）你主要会调哪些

- `--num-episodes-keep`：你要做 5/10/20 的 few-shot 就改它；它决定“从 50 条 demo 里挑出哪几条”。
- `--episode-selection-seed`：决定“挑哪几条 demo”；固定它才能复现。
- `--stride-S`：决定“同一条 demo 里隔几帧取一个训练样本”。
  - stride 大：样本更不重复（训练更省步数），但可能漏掉关键瞬间。
  - stride 小：更稳，但短训时可能浪费在相似帧上。
- `--stage-strategy` + `--num-stages-K`：决定“把一条轨迹分成几段（阶段）”。
  - `time_quantile`：按时间平均切（最稳、最通用）。
  - `change_point_state`：按“状态变化突然变大”找关键转折（可能更强，但更容易被噪声影响）。
- `--stage-balance`：决定“是不是让每个阶段在训练中被看到的次数更接近”。
  - 开启后：抓取/放置这类短阶段不会因为占比小而被采样到太少。
- `--gripper-event-enabled`：是否把“夹爪开合附近的片段”看得更重。
  - 脚本会自动抑制小抖动（先中值滤波，再用分位数阈值只挑最大的变化）。
  - `--gripper-gamma` 越大，越强调夹爪事件附近的窗口。

9.2 训练（`train.py`）你主要会调哪些

- `--sampling-plan-path`：把 plan 接到训练里；不传就还是原 baseline。
- `--num-train-steps`：从 30000 改到 6000/3000 就靠它。
- `--budget-aware-schedule`：短训建议开；它会把“学习率热身（warmup：一开始先用小学习率）/衰减（decay：后面逐步变小）”按总步数自动缩放，避免短训不收敛。
- `--freeze-mode strong`：少样本更稳（冻结更多参数，减少过拟合）。
- norm stats：
  - reuse：不改 `--data.assets.asset-id`
  - recompute：先用 `compute_norm_stats.py` 生成新 asset_id，再在训练里加 `--data.assets.asset-id <new_id>`