

---

# Assignment 2: Single-Task Imitation Learning

EN.601.498 / EN.601.698 · Hands-on Robot Learning · Fall 2026

Weight: 30% · TA practical session: November 5 · In-class demonstration: November 12, 2026

Submission deadline and upload instructions: See Canvas.

## Objective and common requirements

In this assignment, teams will implement a complete workflow for learning a single *task* robot manipulation policy (on the SO-101 arm) via behavioral cloning. In particular, they will collect robot data using teleoperation, train policies on that data, evaluate their performance. Importantly, they will brainstorm and implement approaches that are likely to generalize outside their training data distribution (especially under illumination, background, and potentially camera pose changes).

The assignment has 2 parts: one part is to be completed offline, and the other is to be demonstrated in class. Each component focuses on a different manipulation *task* (implicitly specified by the set of demonstrations).

We recommend that teams that are training robot policies for the very first time use action chunking transformers (ACT) as the method for the first task. It allows for rapid iteration and will get teams a reference solution that can be built upon for the second task. Teams are free to evaluate alternative policy classes, such as diffusion-based policies, provided that the trained policy is still single-task (i.e., has not been pretrained on any other robot manipulation task before).

Do not modify the robot, gripper, wrist camera, or the wrist camera mount. You are allowed to use an RGB-only external camera of your choosing. Upon request, course staff will provide a camera identical to the wrist camera that teams may use as this external RGB camera. Please let the course staff know if this is the case.

## Good practice

Record observations, actions, timestamps, and task labels consistently. We strongly recommend using the LeRobot v3 dataset format for this. Please exercise extra care to document action representation, normalization, control frequency, action prediction horizon (a.k.a. action chunksize), and execution horizon (i.e., number of predicted actions we execute, before moving on to the next action chunk).

## Task 1: Cube placement — take-home evaluation

This task is identical to what teams have looked at in assignment 1. The primary difference is. That we will use behavioral cloning to implement an end-to-end action generation approach, as opposed to a modular sense-plan-act approach. For this task, we require that teams compare at least two learned policies or policy configurations under a clearly defined experimental question. The comparison may study:

- different policy classes, such as ACT versus a diffusion-based policy;
- different prediction chunk sizes;
- different execution horizons; or
- different quantities of training data.

Please discuss with the course staff if in doubt whether your comparison qualifies under this criteria. We take a somewhat broad view of ‘policy configuration’, and are generally supportive of each team finding it’s own path/question to explore as part of this task.

---

The compared systems should differ primarily in the factor being studied. Unless the experimental question specifically concerns one of these factors, all compared systems should use the same demonstration dataset, observation setup, and evaluation distribution. Other training and inference conditions should be kept as comparable as possible.

### Experimental design

For this task, success criteria is identical to assignment 1. The cube needs to be picked up and placed in the tray and the gripper has to let go and move slightly (even if ever-so-slightly) upwards. Partial scoring can be used as a rubric to measure *task progress* (or ‘% task completion’). E.g., moving to the cube, picking up the cube, moving to the tray while holding the picked-up cube, and placing the cube onto the tray all count as fractional success (0.25 each), and if all these 4 substeps are successfully carried out, the policy gets a score of 1 for that trial.

### Rollout/evaluation protocol

(there will be an award for ‘most rigorous evaluation’)

- Each sampled initial configuration (i.e., pose of the cube and pose of the tray) is evaluated across both policy variants
- At least 25 evaluation rollouts (per policy variant)
- For each matched configuration, reproduce the initial condition as consistently as practical. We don’t care too much about being precise to the millimeter, but we care about the comparison across both policies being *fair*.
- Record every evaluation attempt. We require a log of all training demonstrations and evaluations to be submitted.

### STEP analysis (statistical significance testing)

Use the Sequential Testing for Efficient Policy Comparison (STEP) approach to compare your policies. This is the statistical analysis that accompanies the ‘violin plots’ we saw in class when discussing multitask policy learning and large behavior models. STEP tests whether the rollout results support a difference in success rates; the violins show uncertainty in each policy’s success rate.

We need this type of testing because often robot policy evaluations are very small sample sizes, and because of this, we do not know if simply comparing the mean success rates is statistically meaningful/significant. STEP helps determine whether the evidence is sufficient to distinguish the policies.

Start with the [STEP repository and installation instructions](#) and its [quick-start notebook](#). You may use the provided implementation directly; we do not expect teams to reimplement this.

### Running the comparisons

1. **Log per-policy success/failure sequence.** Log a chronological sequence of success/failure labels per policy, with the same initial-configuration IDs aligned across policies. Use 1 (or True) only when the full success criterion above is met, and 0 (or False) otherwise. The 0.25, 0.50, and 0.75 partial scores describe task progress; do not pass them to the binary STEP implementation.
2. **Set the evaluation budget.** Set `n_max` to the planned number of trials per policy before examining the evaluation results. For example, 25 matched configurations with two policies means `n_max = 25` and 50 robot rollouts in total. Retain the default linear risk budget; this determines how the test allocates its error tolerance as trials accumulate.

3. **Compare each pair.** For  $k$  policies, there are  $k(k - 1)/2$  distinct pairs. Use a total false-positive tolerance of 0.10, divided across these pairs: `alpha_pair = 0.10 / (k * (k - 1) / 2)`. Thus, use 0.10 for two policies, or approximately 0.0333 for each of the three comparisons among three policies. This correction limits the chance of making any false separation across the comparisons, under the test’s assumptions. If possible, Use a mirrored test so that either policy can be identified as better. In the current repository, instantiate `MirroredStepTest` with `inference_mode="ranking"`, `alternative=Hypothesis.POLessThanP1`, `alpha=alpha_pair`, and your chosen `n_max`. The ranking option accounts for testing both directions internally; do not divide `alpha_pair` by two again. Use the current repository version if your installed package does not expose this option, and record the version or commit used.
4. **Record the first decision.** With policy A supplied first and B second, `AcceptAlternative` means the test supports B having a higher success rate; `AcceptNull` in this mirrored ranking setting supports A having a higher success rate. `FailToDecide` means a difference has not been established. Record the first decision and the number of trials per policy available at that point. Keep the original rollout order; do not sort or repeatedly reshuffle outcomes to obtain a different decision.
5. **Complete the planned evaluation.** Even if STEP reaches a decision early, finish the planned trials for every policy. Use the full dataset for the success rates, timing summaries, and violins below, while retaining the first STEP decision. If the test is still inconclusive at `n_max`, report that result. Do not increase the budget after seeing an inconclusive result simply to obtain significance.

The statistical model assumes independent trials with a stable success probability for each policy. We recommend randomizing the order of the initial configurations and the policy order within each configuration before evaluation. Matching configurations improves fairness, but shared scene difficulty can correlate policy outcomes; the supplied two-sample test does not explicitly model this dependence. Describe the matched design and any deviations from the model in your report, rather than treating the nominal error tolerance as an unconditional guarantee.

## What to report

For each policy or configuration, report (in a compact tabular form):

- **Successes and failures:** Give the counts, the total number of trials, and the success rate with a 90% Wilson confidence interval. This interval describes uncertainty in one policy’s success rate; comparing interval overlap is not a substitute for the STEP test.
- **Completion time:** For successful trials, measure from the start of policy execution until the full success criterion is met. Report the median and interquartile range, together with the number of successful trials. For unsuccessful trials, report the timeout or earlier termination time and reason. A policy that succeeds quickly only on the easiest trials may still be less useful overall.
- **Task progress and failures:** If you use the partial-scoring rubric above, report it separately from binary success. Explain where failures occur (e.g., reaching the cube, grasping it, transporting it, or releasing it). Relate these observations to the experimental question that motivated your comparison.
- **STEP results:** For every pair, state which policy was favored, or that the comparison was inconclusive, along with the first-decision trial count and test settings. An inconclusive comparison is a valid result; it does not show that the policies are equivalent.

Also include one success-probability posterior violin plot that compares both policies. With  $s$  successes in  $n$  trials and a uniform  $\text{Beta}(1, 1)$  prior, the posterior is  $\text{Beta}(s + 1, n - s + 1)$ . For example, 20 successes in 25 trials gives  $\text{Beta}(21, 6)$ . The violin represents uncertainty about the policy’s underlying success probability, rather than the distribution of its 0/1 rollout outcomes.

---

Annotate the violins with compact-letter-display (CLD) labels obtained from your STEP decisions. Policies that share a letter have not been statistically separated; policies with no shared letter have been separated. For example, labels ‘a’, ‘ab’, and ‘b’ separate the first and third policies, while the middle policy remains unseparated from either.

The following resources provide the implementation and further explanation:

- [Multi-policy comparison notebook](#): use its single-task example to organize the outcome arrays and generate the plots. Adapt its test construction to the mirrored ranking setting above, retain chronological order (`shuffle=False`), and derive CLD labels from the saved pairwise decisions. Use this assignment’s 0.90 overall confidence setting rather than the example’s 0.95.
- [TRI tutorial on statistical policy evaluation](#) and its [CLD and posterior-violin notebook](#): explain why repeated testing needs care and how to interpret the visualizations.
- [Large Behavior Models, Section 4.1](#): describes the evaluation protocol and statistical reporting used in the paper. For this assignment, use the 90% level specified above rather than the paper’s 95% level.

## Task 2: Block stacking — in-class demonstration

For the second task, the challenge is to train a policy outside of class, and then get it to work in-class. This normally represents a significant distribution shift for single-task behavioral cloned policies. We’d like for teams to think about what factors in the external environment are critical to policy performance, and how the environment or model can be adapted to ensure better out-of-distribution performance. We allow teams to change the robot workspace (e.g., by building a box / separators, or bespoke camera mounts for consistency, etc.). We will offer awards (and extra credit) to teams that try to keep the changes to external environment minimal. (But there won’t be any penalization for teams that do choose to tweak the robot workspace).

The task for this part of the assignment is to **train a single-task policy to retrieve blocks from a box or tray and construct a freestanding stack**. The minimum target is to get a stack of three blocks; but teams are encouraged to attempt taller stacks. The robot must place the blocks, let go, and withdraw without supporting the stack. We will only count the blocks that remain stacked after the robot arm withdraws. We expect some robustness to initial configurations of the blocks. However, the tower may always be constructed at a fixed location relative to the robot.

We recommend reusing the data collection, training, and deployment code that you developed for Task 1 of this assignment, then collecting demonstrations for stacking and training its policy. Stacking introduces an additional challenge: the placement target changes as the stack grows, and small placement errors can accumulate. Your demonstrations should cover these different stages, rather than repeatedly showing only the first block placement.

### Workspace and generalization

Describe the setup in which your policy is expected to work: the initial block arrangement, the position of the source box or tray, the stacking region, and any workspace fixtures or constraints. Include a photograph or sketch so that the setup can be reproduced in class. If you use an external camera, document its placement and any calibration procedure. The earlier restrictions on modifying the robot and wrist camera still apply.

A policy that works on your desk/home may fail when the background or illumination changes in the classroom. Think about this while collecting data and training, rather than only on the day of the demonstration. For example, teams may collect demonstrations under varied lighting or backgrounds, or use appropriate image augmentation during training. Explain which variation your approach is intended to handle and test whether it actually helps. Image augmentation alone does not guarantee robustness to changes in camera pose.

---

We recommend testing a few controlled changes separately (e.g., change the background while retaining the same block arrangement), and recording both successful and failed attempts. Do not infer robustness from one successful run. Keep the wrist camera and its mount unchanged; any camera-placement experiments must respect the common requirements above.

### In-class demonstration and awards

The November 5 TA practical session will cover single-task learning practices and provide an opportunity to collect additional classroom demonstrations. Use this session to check whether your setup transfers to the lecture room. Any fixtures, external-camera placement, or other controlled conditions on which your policy depends must be documented and reproducible for the November 12 demonstration.

Awards will recognize the **most performant demonstration, the tallest stable stack, and the fastest successful three-block stack**. Performance includes reliability across attempts (not just the *best* attempt).

**Please note:** Course staff will ask every team member questions to test their understanding about the complete system, including data collection, policy architecture, training, inference, control, and failure cases. Please make sure that each member can explain why the team made its design choices and what the system can and cannot do.

### Submission and assessment

Each team will submit 1 zip file, that has multiple subfolders, organized by task:

- **Task 1:** Include the demonstrations, all evaluated checkpoints and configurations, and logs plus videos of every evaluation rollout. Provide a short table or manifest linking each rollout’s episode ID to its policy, initial configuration, success/failure label, any partial score, and timing. Include the analysis code needed to reproduce the statistical results and plots. Your **write-up (at most one page)** should explain the comparison question, design choices, results, and limitations.
- **Task 2:** Include the demonstrations, deployed checkpoint and configuration, code, and representative videos of successful and failed attempts. Your **system description (at most one page)** should explain the workspace, how the policy was trained and tested, its sensitivity to changes in conditions, and the failures you observed.
- **Both tasks:** Include reproduction instructions and software versions, together with a separate short contribution statement describing what each member did. Disclose which AI tools you used and for which parts of the work.

Use the recommended [LeRobotDataset v3 format](#) where possible. If your rollout dataset already contains the required logs and camera videos, you do not need to submit duplicate copies; make the episode-to-policy mapping clear. If you use another data format, include enough documentation or loading code for course staff to read the observations, actions, and timestamps.

Assessment will consider implementation, whether the experimental comparison is fair, the quality of the failure analysis, demonstration performance, and each team member’s understanding. AI tools may be used liberally, but teams must check their outputs and understand everything they submit. Datasets, checkpoints, videos, logs, and analysis code are submitted separately (i.e., they don’t count toward the two one-pager write-up limits).