STATS 607 Final Project Report: Accelerating Off-Policy Evaluation Computation in Confounded POMDPs

Ziheng Wei

December 2025

1 Motivation

Off-policy evaluation (OPE) estimates the value of a target policy using data generated by a different behavior policy. In partially observable MDPs (POMDPs) with confounding, naive estimators can be biased. NeurIPS 2022 paper ***Off-policy evaluation for episodic partially observable markov decision processes under non-parametric models*** proposed a value-based approach, which solves nonparametric instrumental variable (NPIV) problems at each step t by RKHS-based minimax estimation. This approach has been a principled method for policy evaluation and policy learning for POMDPs under confounding. The reference NeurIPS 2022 simulation code is correct but becomes slow (often took 2 to 3 hours) and fragile for parameter sweeps over horizon length, sample size, and random seeds.

This project aims to make the simulator and estimator pipeline fast, stable and easy to run on common environments. The main beneficiaries are (i) researchers who study OPE or OPL in POMDPs, and (ii) who study NPIV problems in Reinforcement Learning.

2 Project Description

2.1 Problem setting.

The environment is a confounded, partially observable, continuous-state simulator. The target policy is evaluated using offline trajectories collected under a different behavior mechanism.

2.2 Implementation Improvements (Compared to the Original Codebase)

The estimator and the sequential backward recursion are preserved. All changes are engineering-focused: updated interfaces, faster execution, improved numerical stability, and a clearer, reproducible workflow.

1) API modernization and codebase portability

- **Gymnasium-compatible environment interface.** The original code relied on a locally cloned `gym/` directory and legacy assumptions about `reset`/`step` return values. The updated version uses `gymnasium` directly and makes the data collectors compatible with the modern API, which removes path-dependent behavior and reduces brittle glue code.
- **Modern PyTorch usage and device handling.** The code is reorganized to use current `torch.linalg` routines, consistent `dtype`/device placement, and safer scalar extraction (e.g.,

avoiding legacy patterns that can silently create device copies or deprecated behaviors). This improves maintainability and reduces accidental slowdowns.

2) Runtime acceleration

- **Vectorized trajectory collection.** The updated collector preallocates arrays and performs batch conversion to tensors, reducing per-step allocations and interpreter overhead.
- **CPU parallelization over seeds.** For CPU runs, independent random seeds are parallelized within each (T, n) job using multiprocessing. GPU runs remain single-process by default to avoid contention on a single device.
- **Cached Monte Carlo true value.** The Monte Carlo estimate of the target policy value can be cached under `.cache_pomdp/`, so repeated sweeps do not recompute the same expensive true-value estimate.

3) Numerical stability improvements in RKHS

- **PSD-aware matrix operations.** The RKHS solver avoids expensive or fragile matrix square-root routines by preferring PSD-aware eigen-decompositions (`eigh`) with explicit diagonal jitter when needed, which is more stable for nearly singular Gram matrices.
- **Stable linear system solves.** Linear solves of $(A + \lambda I)x = b$ prioritize numerically stable routines (e.g., Cholesky when applicable) and fall back to least squares only when necessary.
- **Explicit symmetry and regularization.** Kernel matrices are symmetrized when appropriate and regularized with small diagonal jitter to control numerical issues at larger n.

4) Kernel computation and cross-validation efficiency

- **Distance matrix reuse.** For RBF kernels, squared-distance matrices are reused across hyperparameter grids so that multiple γ values are evaluated via `exp(-`γD^2`)` without repeating expensive `cdist` calls.
- **Reduced redundant linear algebra in CV.** In cross-validation loops, computations that only differ by ridge parameters are reorganized to reduce repeated work (e.g., reusing shared components whenever possible rather than solving from scratch for each hyperparameter).
- **Nyström approximation for acceleration.** The approximation pipeline is implemented in a way that better supports caching and stable solves, improving both speed and robustness.

5) Reproducible workflow and clearer project organization

2.3 Which technique I used

- **Simulation-study workflow.**
- **Parallelization.**
- **Vectorization and broadcasting.**
- **Numerical stability.**

3 Results or Demonstration

How to run (demo). The following command should complete quickly and produces a per-seed CSV:

```
python ContSimuOffPolicy.py --T 2 --n 64 --epsilon 0.2 --device cuda:0 ^
  --mc-episodes 200 --eval-initial 200 --n-sims 2 ^
  --out results\tables\demo.csv
```

The CSV contains one row per seed with columns: `episode_len, samp_size, epsilon, seed, abs_error, value_hat, true_value, fit_sec`.

Sweeps and figures. Two standard sweeps are supported:

- **Mode A (horizon sweep):** fix $n = 512$ and vary T.
- **Mode B (sample-size sweep):** fix $T \in \{2, 4, 6\}$ and vary n.

Run the sweep and then plot:

```
set POMDP_N_WORKERS=4 && python scripts\run_grid.py --mode AB --device cpu --epsilon 0.2
    --mc-episodes 50000 --eval-initial 10000 --n-sims 100 --out-root results --quiet

python scripts\plot_results.py --out-root results
```

This generates:

- `results/tables/ResultA.csv`, `results/tables/ResultB.csv`
- `results/figures/fig_modeA_4panel.png`, `results/figures/fig_modeB_4panel.png`

Example output. Table 1 shows an example demo output (two seeds) from a small run.

Table 1: Example demo output (per-seed).

T	n	ϵ	seed	abs_error	value_hat	true_value	fit_sec
2	64	0.2	0	0.0963	1.1085	1.2049	0.6314
2	64	0.2	1	0.0739	1.1310	1.2049	0.4421

Figures: Sweep results. Please see in the last page.

4 Lessons Learned

Challenges. The main challenges were (i) parallelization on GPU and CPU, (ii) keeping RKHS computations numerically stable under large n and cross-validation grids, (iii) reducing calls on RKHS solves as many as possible, (iv) vectorization of NPIV problems.

What changed because of the course. I shifted from optimizing isolated functions to designing an end-to-end, reproducible simulation pipeline. I also started treating performance and stability as first-class constraints (vectorized data paths, parallelization, caching, and stable linear algebra). Actually this really helped me a lot in my current research project which is similar to the final project. I used parallelization and vectorization technique, which saved a lot of time in running the code.

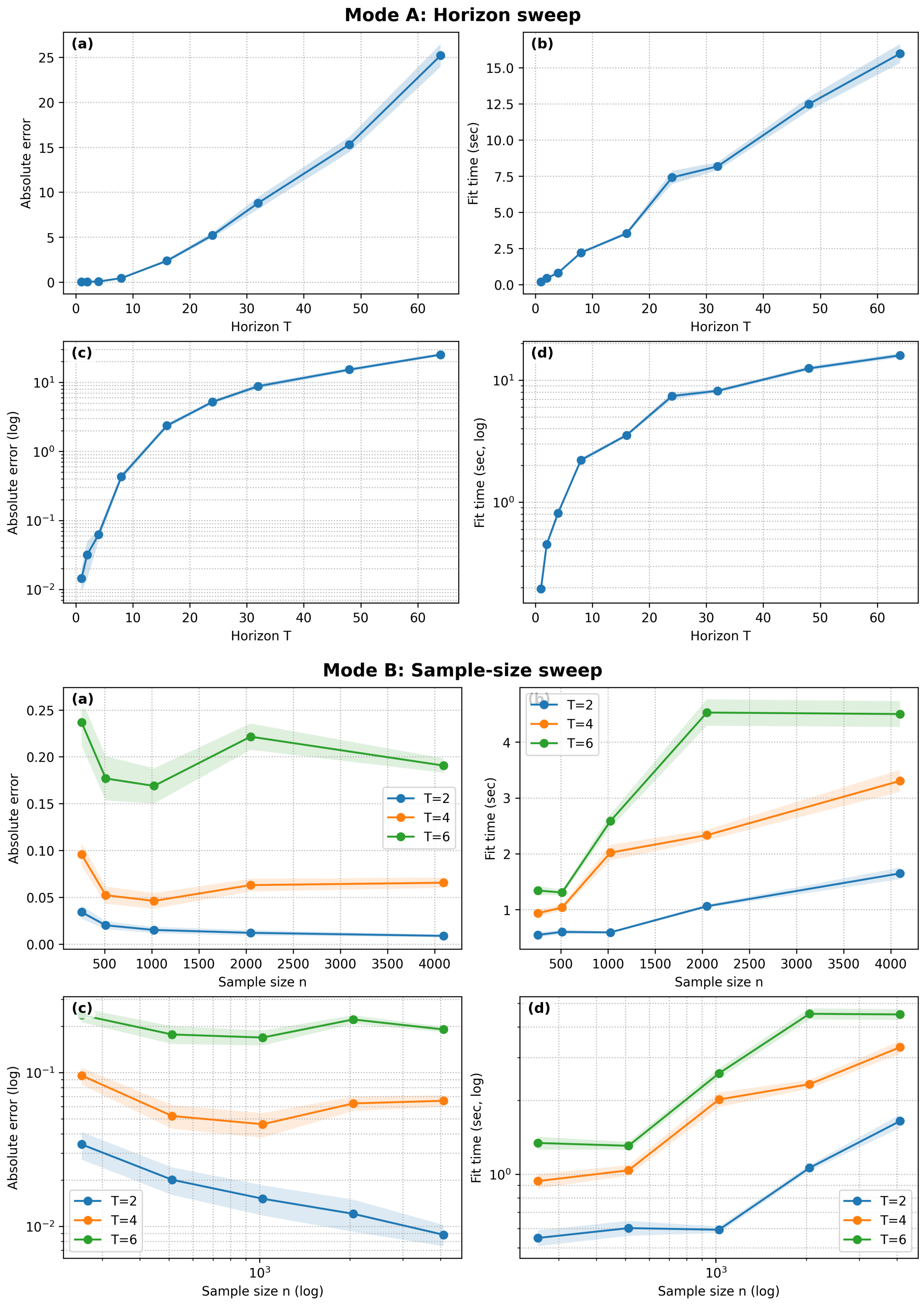

Figure 1: Mode A and Mode B sweep summaries.