ChatGPT

Performance Diagnosis of PPO Policy vs Random Baseline

Overview of Training Metrics and Observations

Aspect	PPO Policy Observation	Implication
Reward Trajectory	**Plateaued at negative values**. Mean episodic reward stabilizes around **-0.2 to -0.3**, with high variance (std ~1.5). Rewards occasionally hit a floor near **-6** (no large positive spikes)【43†】. No upward trend in final stages – if anything, rewards became **more negative** as throughput increased.	The agent **never achieved positive reward** or outperformed baseline on the reward scale. It suggests the policy converged to a **suboptimal trade-off**: as it pushed throughput to maximum, heavy latency penalties drove the net reward down. The lack of improvement (flat/negative trajectory) indicates the policy **stalled** in a local optimum where increasing throughput further only incurs more penalty.
Entropy & Exploration	**High entropy persisted.** The policy did **not collapse to a single action** – by the end it was choosing each replica ~25% (uniformly)【42†】. This corresponds to an entropy near the theoretical maximum (≈1.39 for 4 actions). Entropy coefficient was set low (0.02), and indeed the agent's action distribution remained **highly stochastic** rather than becoming deterministic.	The agent essentially behaves like a **random policy**. The near-uniform action distribution means it did not learn a more refined scheduling strategy beyond evenly spreading load. Persistently high entropy implies **insufficient exploitation** – the policy never "locks in" an improved policy, likely because it **never found a clearly better-than-random action pattern**. The very low entropy regularization may have actually caused early convergence to a shallow optimum (uniform random) and then maintained it (since a uniform policy already satisfies exploration to some degree, the agent had no pressure to change).

Aspect	PPO Policy Observation	Implication
KL Divergence & Updates	**Stable/low policy updates** (inferred). The policy distribution barely shifted in later training – action probabilities stayed roughly constant. No signs of policy oscillation or divergence. (Direct KL metrics not logged, but the stagnation of action distro and reward suggests KL remained small in updates.)	The training was **too conservative** in updating the policy. PPO likely kept the policy close to its prior (low KL), which prevented exploring fundamentally different strategies once it had converged to the uniform distribution. This stability indicates the agent got **stuck in a local optimum** early and never escaped. With the policy not changing much, it wasn't trying new scheduling approaches that might outperform random.
Action Distribution & Imbalance	**Balanced across replicas.** Initial training showed one replica overused (e.g. 0 got only ~16% vs others ~27% in early steps)【54†】, but by the end the agent sends ~25% of requests to each replica (0:145,1:152,2:144,3:147 out of 588 final actions)【42†】. The **direct imbalance penalty** is essentially minimized – average queue-length std ≈0.03 (nearly zero)【45†】. No single replica is oversaturated in the final policy.	**Load imbalance was effectively eliminated**, which was a goal of the reward design. However, achieving perfect balance did **not yield performance gains** over random – the random policy already distributes load fairly well on average. The agent did not go beyond uniform balancing to something smarter. In fact, the heavy emphasis on balance likely **dominated the policy's focus**, causing it to prioritize equal distribution above all else. This may have come at the cost of other objectives (e.g. it might ignore opportunities to improve latency or throughput if they involve slight imbalance).

Aspect	PPO Policy Observation	Implication
Reward Component Skew	**Latency penalties dominate reward.** By end of training, throughput is maxed (~3.15 vs arrival ~3.0) but latency is very high (mean ~9.42, p95≈9.42) – leading to large negative **absolute latency penalties**. The **imbalance penalty** component is near zero (balanced load), and the **throughput reward** is capped by arrival rate. Any small logistic/Delta rewards are dwarfed by the latency term. Net immediate rewards are negative in virtually every step (no positive reward region).	The reward function is **skewed toward penalties**, especially latency. The agent finds itself in a scenario where every decision yields a negative reward (because full utilization inherently causes latency explosion). With throughput limited by the environment and imbalance solved, the agent cannot increase reward except by reducing latency – but latency can only drop by processing fewer tasks (which would reduce throughput reward). This trade-off wasn't properly balanced in the reward design. In effect, the **penalty for latency is so severe at capacity that the optimal policy in terms of reward is to stay just at capacity** – which the random policy already does. The agent had no incentive or means to improve overall reward significantly, resulting in a "**dead" reward signal** once it achieved uniform distribution.
Latency and Tail Behavior	**High latency persisted.** Final average latency ~9.415 with virtually no variation (max ~9.425)【46†】, indicating the system is consistently at saturation latency. The policy did not reduce tail latency; it basically hit the latency ceiling imposed by the workload. (Random scheduling under the same load would also see ~9.4 latency on average – no worse in logs.)	The PPO policy **failed to improve latency** relative to random. Despite enhanced state features (which presumably include queue lengths, etc.), the agent did not learn to, for example, **send jobs to the least-loaded replica to cut queueing delay** – the latency remained as high as the worst-case baseline. This suggests a **missed opportunity in dynamic scheduling**: the policy wasn't leveraging state information to reduce queuing, likely because the reward didn't strongly reward slight latency improvements (and any such improvement would be marginal once at full load). In effect, the agent's actions had **no impact on latency** – it ended up matching the baseline latency curve.

Likely Failure Modes

- **Over-Conservative Exploration (Entropy Too Low)** – The entropy bonus was drastically reduced (0.25→0.02). As a result, the agent likely **converged too quickly** to a suboptimal policy. Once it found a "reasonable" strategy (uniform load spreading), there was little entropy pressure to explore further. The final high entropy of the policy is deceiving – it's not deliberate exploration,

but rather the agent staying in a high-entropy uniform strategy. Essentially, the policy became **"stuck being random."** Any potential better strategies (e.g. intelligently favoring certain replicas under certain conditions) were never properly explored.

- **Dead Reward Signal at High Load** – The reward function, even after "unclipping," still led to mostly negative rewards in the high-load regime. The agent never sees a clear reward **differential** for a better-than-baseline action because even "optimal" performance yields negative reward (latency penalties cancel out throughput gains). This can lead to **learning stagnation** – the policy has no gradient push toward improvement if every action seems bad. In other words, once the agent achieved decent load balance (removing the huge imbalance penalty it initially suffered), the remaining reward landscape was flat: all actions resulted in similarly poor total reward due to inherent latency at capacity. This **lack of a positive reward signal** for further improvements means the agent had no guidance to surpass the random policy.

- **Objective Imbalance – Over-Penalized Imbalance vs Throughput** – The 10× increase in imbalance penalty weight succeeded in enforcing balance, but it likely **overcorrected** the policy's priorities. The agent almost certainly learned "avoid any imbalance at all costs" very early (we see `direct_imbalance_penalty` hammered to near zero). Once that was achieved, the policy may have been **overly cautious** about ever deviating from equal distribution, even when transient imbalance could have, for example, allowed one replica to batch process requests faster. In sum, the reward function's skew made the policy **too risk-averse**: it treats any uneven load as worse than potential throughput gains, mirroring a strict round-robin behavior. This prevented learning of strategies that momentarily imbalance load to reduce overall latency or increase batching.

- **Lack of Dynamic Scheduling Behavior** – Despite adding state features (queue lengths, etc.), the learned policy does **not appear to use them** in a meaningful way. A truly intelligent scheduler might, for instance, send new requests to the currently least-busy replica (to minimize queueing) – this would show up as the action distribution varying in response to state. Instead, the PPO policy's action probabilities are basically static 25/25/25/25, regardless of state. This suggests a **failure to exploit the richer state information**. Likely causes are the domination of a simple strategy (uniform distribution) and insufficient incentive to fine-tune decisions based on state (since the reward didn't noticeably improve when doing so). In effect, the policy behaves "memoryless" and random – a hallmark of a failed curriculum or training where the agent never graduated to using the nuanced state inputs.

- **Curriculum Gap or Early Stopping** – The training used a curriculum (gradually increasing load or difficulty). It's possible the agent performed well in easier scenarios (where latency wasn't as extreme) but **hit a wall in the final scenario**. For example, it may have learned to balance load (solving the easier reward issues) but the curriculum **did not prepare it to optimize latency at max throughput**. If the final jump to full load was too abrupt or if training didn't continue long enough at full difficulty, the policy may have **plateaued** at a baseline strategy. The absence of temperature "pulsing" in late training (no exploration boosts were triggered) implies the training loop believed improvement was happening (or stagnation detection failed) – potentially a curriculum tuning issue. The result is a policy that **never surpassed the round-robin baseline** after curriculum ramp-up.

- **Possible Implementation Issues** – While less likely given the data, one should not ignore the chance of subtle bugs: e.g., **action masking or reward scaling errors**. If, say, the agent wasn't truly free to choose some actions (or a replica selection was effectively ignored due to a masking bug), it might reduce the space of strategies (though uniform usage suggests it could select all).

Similarly, if reward “unclipping” was not properly propagated (e.g. logging shows rewards outside the ±4 clip, hinting at some mismatch), the agent might still be effectively getting clipped gradients. These would further hinder learning. No blatant bug is evident, but it's worth a double-check given the puzzling lack of improvement.

Actionable Next Steps

1. **Rebalance the Reward Function:** Adjust the reward weights to ensure no single penalty dominates. Specifically, **reduce the imbalance penalty weight** (e.g. from 1.0 down closer to 0.2–0.5) so the agent doesn’t treat tiny load differences as catastrophic. Simultaneously, consider boosting the positive reward for throughput or low latency. The goal is to create a reward landscape where the agent is encouraged to explore strategies that **improve latency or throughput**, even if they introduce a bit of imbalance. For example, if batching on one replica could increase throughput, the reward should clearly reflect that benefit over a perfectly balanced but lower-throughput scenario. A more balanced reward will guide the agent toward better trade-offs rather than rigidly enforcing one metric.

2. **Improve Exploration Schedule:** The agent likely got stuck due to insufficient exploration later in training. Revisit the entropy hyperparameters and temperature schedule:

3. **Increase `entropy_coef` or use decay scheduling** – e.g. start around 0.1 instead of 0.02 and decay it slowly rather than cutting it so low from the start. This will prevent premature convergence and encourage continued strategy search.
4. **Tune the stagnation detection for temperature pulses.** It appears no pulses occurred when the policy plateaued. Ensure that the stagnation criteria (e.g. improvement threshold or window) are sensitive enough to trigger a temperature pulse when the policy is flatlining. If the criteria were too strict, the agent might never get that exploration boost. A timely pulse (temporary high-temperature exploration) in the final curriculum phase might kick the policy out of the local optimum.

5. **Use larger batch or more parallel environments** to inject variability – sometimes training with more diverse experiences can prevent the agent from getting tunnel-vision on one strategy.

6. **Reward Signal Shaping:** Introduce mechanisms to avoid a “dead zone” in rewards:

7. You might implement a **baseline or reference reward** (e.g., reward the agent relative to a known baseline like random/round-robin performance). If every action yields negative absolute reward, the agent only learns “less bad” vs “more bad.” By framing reward as an advantage over a baseline (even a moving average of past performance), you give positive feedback for outperforming what it’s already doing. This could help it climb out of the flat region.
8. Ensure that **reward clipping** truly is removed or set high enough. The logs showed rewards below -4 (e.g. -6.11【43†】), which might indicate either the clipping wasn’t in effect or was applied per-component not on total. Double-check that the agent isn’t inadvertently saturating any reward bounds during optimization – it should feel the gradient of making latency slightly better, even if overall reward is negative.

9. If latency at full load is always huge, consider adding a **small positive reward for reductions in latency** (or negative reward for increases) on a relative scale. For instance, reward improvements in p95 latency from one episode to the next. This could amplify the signal to the agent when a policy change actually reduces tail latency, which absolute latency penalty alone did not achieve (since it was always high).

10. **Leverage State Features in Policy:** The agent currently isn't utilizing the rich state information. To encourage this:

11. **Audit the observation space** to ensure features like queue lengths, service times, etc. are correctly normalized and impactful. If the agent sees only tiny differences (e.g., queue length 5 vs 6 might be lost in noise), it may ignore them. Scaling these inputs or using an architecture that emphasizes them (attention or a separate value head for balance) could help the policy differentiate states.
12. **Incorporate state-dependent rewards** as a training signal. For example, directly reward negative queue length variance or waiting time reduction. This could be risky (might overlap with penalties), but done carefully it could teach the agent why and when to favor one replica. Essentially, make it explicit in the reward when choosing the currently fastest (least busy) replica leads to better outcomes (shorter latency).

13. Ensure the policy network is sufficiently expressive (the **"enhanced architecture"** should be verified). Perhaps add an LSTM or attention mechanism if sequence patterns matter (though for load balancing a feed-forward might suffice). A more expressive model might discover non-trivial scheduling patterns (like alternate who gets the next request depending on who is busiest).

14. **Extended Training and Curriculum Tuning:** It may simply be that the policy needs more time or a gentler curve in the hardest scenario:

15. **Continue training in the final load condition** for more iterations with higher entropy or occasional perturbations. Monitor if the average reward or any metric budges. Sometimes the solution is to give the agent more experience at the challenging task once the easier objectives (balance) are learned.
16. Re-examine the **curriculum progression** – if the jump to full QPS was too sudden, insert an intermediate stage or ramp up more slowly so the agent has time to adjust its policy. For instance, gradually increase QPS from, say, 2.0 to 3.0 over several training epochs, while continuing to emphasize latency improvements.

17. **Evaluate intermediate policies** (e.g., from earlier in training or with different entropy settings) against the baseline. It's possible the agent had a better policy at some point (e.g., slightly imbalanced but better throughput) but that policy was "trained away" because the reward function penalized it. Identifying such cases can guide reward reweighting.

18. **Validate Against Random/Round-Robin Explicitly:** Incorporate the baseline comparison into training or validation. If possible, periodically measure the agent's performance (latency, throughput, imbalance) against a random or round-robin policy. If the agent is not beating them, that's a red flag during training. You can use this as an additional feedback metric (not directly to the agent, but for deciding when to trigger exploration pulses or when to adjust hyperparameters). For example, if PPO < Random in metrics for N episodes, increase entropy or adjust rewards. This ensures the agent doesn't spend thousands of steps reinforcing a subpar strategy.

By implementing the above changes, we expect the PPO agent to **break out of its current plateau**. In practice, we want it to maintain the good load balancing it achieved, but also start leveraging its state awareness to reduce latency and maybe slightly oversubscribe the best replica to boost throughput when beneficial. The priority is to provide a clear learning signal for those improvements: more balanced reward weights and sustained exploration should lead the policy to finally **surpass the random policy** on all key metrics (latency, throughput, imbalance).