Certainly! Here's an illustration and explanation of the **PPO architecture** and how losses and rewards are incorporated, tailored for your JupyterLab work in the Generative AI Advanced Fine-Tuning for LLMs course.

**PPO Architecture Overview**

1. **Policy Network (Actor):**
   * The language model (e.g., GPT-2) acts as the policy network.
   * Given an input query, it generates responses (actions) by sampling tokens based on learned probabilities.
2. **Value Network (Critic):**
   * Often implemented as a separate head on the same model or a separate network.
   * Estimates the expected cumulative reward (value) for a given input state (query).
3. **Environment & Rewards:**
   * The environment is the task setting where the model generates text.
   * Rewards are numerical feedback signals evaluating the quality of generated responses (e.g., sentiment score).

**How PPO Uses Loss and Rewards**

**Step-by-step during training (ppo\_trainer.step()):**

1. **Generate responses** from the current policy for input queries.
2. **Calculate rewards** for these responses using a reward function (e.g., sentiment analysis pipeline).
3. **Compute advantage estimates:**
   * Advantage = Reward + (discounted) future value - current value estimate
   * It measures how much better an action (response) is compared to the expected value.
4. **Policy Loss (Clipped Objective):**
   * Calculate the ratio of new policy probability to old policy probability for the taken actions.
   * Use a clipped surrogate objective to prevent large policy updates:  
     [ L^{CLIP}(\theta) = \mathbb{E} \left[ \min \left( r(\theta) \hat{A}, \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A} \right) \right] ]
   * Here, (r(\theta)) is the probability ratio, (\hat{A}) is the advantage, and (\epsilon) is a small threshold.
5. **Value Loss:**
   * Mean squared error between predicted values and actual rewards (or returns).
6. **Total Loss:**
   * Combination of policy loss and value loss (plus optional entropy bonus for exploration).
7. **Backpropagation:**
   * Compute gradients of total loss w.r.t. model parameters.
   * Update parameters with gradient descent.

**Visual Summary**

Input Query --> Policy Network --> Generated Response --> Reward Function --> Reward Score

↘

Value Network --> Value Estimate

Use Reward & Value Estimate --> Calculate Advantage --> Compute PPO Losses --> Update Model Parameters

**In your JupyterLab environment**

* The ppo\_trainer.step(query\_tensors, response\_tensors, rewards) method encapsulates all these steps.
* It updates the model weights to improve policy (response generation) and value estimation based on rewards.
* You can monitor training progress via returned statistics like policy loss, value loss, and average reward.

If you'd like, I can help you with code snippets to visualize these losses or interpret training logs in your notebook.

[See related lab content](https://www.ibm.com/cloud/learn)

Would you like me to find similar content?