

# GPT-2 with Q-Head: Technical Overview

## 1 Abstract

This document describes a method for augmenting GPT-2 with a Q-head that predicts expected future returns for each possible next token. The approach combines standard language modeling with reinforcement learning concepts, enabling the model to learn both token prediction and value estimation simultaneously.

## 2 Main Idea

The core insight is to treat autoregressive language generation as a sequential decision-making process. At each position  $t$  in a sequence, the model must choose the next token  $a$  from the vocabulary  $\mathcal{V}$ . We augment GPT-2 with a second output head that predicts  $Q(s_t, a)$  — the expected discounted future return if action  $a$  is taken in state  $s_t$ .

### 2.1 Architecture Overview

- **Backbone:** GPT-2 transformer producing hidden states  $h_t$  of dimension  $H$
- **LM Head:** Linear projection  $h_t \rightarrow$  logits over vocabulary (standard next-token prediction)
- **Q Head:** Linear projection  $h_t \rightarrow$  Q-values over vocabulary (value prediction for each action)

The Q-head outputs a  $|\mathcal{V}|$ -dimensional vector at each position, where each component  $Q(s_t, a)$  represents the expected future return if token  $a$  is selected as the next token. This allows querying the value of any action without needing to actually sample it.

## 3 Objective Function

The training objective combines two loss terms:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{LM}} + \lambda \cdot \mathcal{L}_Q \quad (1)$$

where  $\lambda$  (`q_weight`) controls the relative importance of the Q-learning objective.

### 3.1 Language Modeling Loss ( $\mathcal{L}_{\text{LM}}$ )

Standard causal language modeling loss using cross-entropy. For a sequence of tokens  $(x_1, x_2, \dots, x_T)$ :

$$\mathcal{L}_{\text{LM}} = - \sum_{t=1}^{T-1} \log P(x_{t+1} | x_1, \dots, x_t) \quad (2)$$

This is computed using shifted logits: predictions at positions  $0..T-2$  are compared against ground truth tokens at positions  $1..T-1$ . Padding positions are masked out.

### 3.2 Q-Value Loss ( $\mathcal{L}_Q$ )

The Q-head is trained using mean squared error between predicted Q-values and return targets. For the observed action  $a_t$  (the actual next token):

$$\mathcal{L}_Q = \frac{1}{N} \sum_t (Q(s_t, a_t) - G_t)^2 \quad (3)$$

where  $G_t$  is the return target from position  $t+1$  onward. The choice of return target is configurable (see Section 4).

## 4 Computing Return Targets

A key challenge is handling the context window boundary. The sequence doesn't truly end at position  $L$ —it's an arbitrary truncation. Treating it as a terminal state (future value = 0) would bias Q-estimates downward near the boundary.

### 4.1 Bootstrap Value at Context Boundary

We bootstrap using the expected Q-value under the current policy at the last position:

$$V_{\text{bootstrap}} = \mathbb{E}_{a \sim \pi}[Q(s_{L-1}, a)] = \sum_a \pi(a|s_{L-1}) \cdot Q(s_{L-1}, a) \quad (4)$$

where  $\pi(a|s) = \text{softmax}(\text{logits})$  is the policy distribution from the LM head.

### 4.2 Monte Carlo Returns (Default)

The default return target uses full Monte Carlo returns with bootstrapping:

$$G_t^{\text{MC}} = \sum_{k=t+1}^{L-1} \gamma^{k-(t+1)} \cdot r_k + \gamma^{L-1-t} \cdot V_{\text{bootstrap}} \quad (5)$$

where  $\gamma$  is the discount factor (default 0.99) and  $r_k$  is the reward at position  $k$ .

### 4.3 GAE $\lambda$ -Returns (Optional)

As an alternative to Monte Carlo returns, we implement **Generalized Advantage Estimation (GAE)**  $\lambda$ -returns, which interpolate between TD(0) and Monte Carlo estimates.

The  $\lambda$ -return is a weighted combination of  $n$ -step returns:

$$G_t^\lambda = (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t^{(T-t)} \quad (6)$$

where the  $n$ -step return is:

$$G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V(s_{t+n+1}) \quad (7)$$

### 4.3.1 Recursive GAE Formulation

Equivalently, GAE can be computed recursively using TD errors:

$$\delta_t = r_{t+1} + \gamma V(s_{t+2}) - V(s_{t+1}) \quad (\text{TD error}) \quad (8)$$

$$A_t^{\text{GAE}} = \sum_{k=0}^{T-t-1} (\gamma\lambda)^k \delta_{t+k} \quad (\text{GAE advantage}) \quad (9)$$

$$G_t^\lambda = A_t^{\text{GAE}} + V(s_{t+1}) \quad (\lambda\text{-return target}) \quad (10)$$

### 4.3.2 Value Function Estimation

For GAE, we need value estimates  $V(s_t)$  at each position. We compute these as the expected Q-value under the policy:

$$V(s_t) = \mathbb{E}_{a \sim \pi}[Q(s_t, a)] = \sum_a \pi(a|s_t) \cdot Q(s_t, a) \quad (11)$$

### 4.3.3 Bias-Variance Trade-off

The  $\lambda$  parameter controls the bias-variance trade-off:

- $\lambda = 0$ : TD(0) — low variance, high bias (one-step bootstrap)
- $\lambda = 1$ : Monte Carlo — high variance, low bias (full returns)
- $\lambda \in (0, 1)$ : Interpolation (typical values: 0.95–0.99)

### 4.3.4 Implementation

GAE is enabled via the `--gae_lambda` flag:

```
python train.py --gae_lambda 0.95 --steps 50000 --wandb
```

## 5 Reward Specification

### 5.1 Zero Rewards (Default for Streaming)

For streaming datasets, rewards default to zero:  $r_t = 0$ . The Q-head learns to predict expected future “value” based purely on context, which with zero rewards becomes an estimate of discounted future bootstrap values.

### 5.2 Log-Probability Rewards

When `--use_lm_rewards` is enabled, rewards are computed as the log-probability of the next token under a reference model:

$$r_t = \log P_{\text{ref}}(x_{t+1} | x_1, \dots, x_t) \quad (12)$$

This encourages the Q-head to predict expected “fluency” as measured by the reference language model.

## 6 Q-Tilted Sampling

At inference time, we can use the Q-values to bias sampling toward higher-value actions. The **Q-tilted policy** is:

$$\pi'(a|s) \propto \pi(a|s) \cdot \exp\left(\frac{Q(s, a)}{\beta}\right) \quad (13)$$

Equivalently, in log-space:

$$\text{logits}'(a) = \text{logits}(a) + \frac{Q(s, a)}{\beta} \quad (14)$$

where  $\beta$  is the inverse temperature:

- $\beta \rightarrow \infty$ : Original policy  $\pi$  (ignore Q-values)
- $\beta \rightarrow 0$ : Greedy w.r.t. Q-values (pure exploitation)
- Typical:  $\beta \in [0.5, 2.0]$

## 7 Key Properties

1. **On-Policy Supervision:** The Q-head is only supervised for the actually observed next token (the action that was taken). This is a form of on-policy learning where we learn Q-values along the trajectory defined by the training data.
2. **Vocabulary-Sized Q-Output:** Unlike scalar value functions  $V(s)$ , the Q-head outputs a value for every possible action. This enables action selection via  $\arg \max_a Q(s, a)$  at inference time.
3. **Shared Representations:** Both heads share the transformer backbone, allowing the Q-head to benefit from the rich representations learned for language modeling.
4. **Flexible Return Estimation:** Choice between Monte Carlo returns (lower bias) and GAE  $\lambda$ -returns (tunable bias-variance).
5. **Flexible Reward Specification:** Rewards can encode any per-token signal — correctness, style, safety scores, or learned reward models.

## 8 Summary

The GPT2-with-Q-head architecture demonstrates how reinforcement learning concepts can be integrated into language models. The combined objective:

$$\mathcal{L} = \mathcal{L}_{\text{LM}} + \lambda \cdot \mathcal{L}_Q \quad (15)$$

trains the model to simultaneously predict next tokens (via cross-entropy) and estimate expected future returns (via MSE on return targets). With GAE  $\lambda$ -returns, users can tune the bias-variance trade-off for Q-learning. This provides a foundation for value-guided text generation and reinforcement learning from human or automated feedback.

## 9 Command-Line Reference

Key training parameters:

<code>--gamma</code>	Discount factor (default: 0.99)
<code>--q_weight</code>	Weight for Q-loss term (default: 1.0)
<code>--gae_lambda</code>	GAE lambda (None=MC, 0=TD(0), 0.95=typical)
<code>--use_lm_rewards</code>	Use log-prob rewards from reference model
<code>--seed</code>	Random seed for reproducibility
<code>--save_interval</code>	Save checkpoints every N steps
<code>--save_dir</code>	Directory to save checkpoints