DSC291: Advanced Statistical Natural Language Processing

Text Generation

Zhiting Hu

Lecture 14, May 12, 2022

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

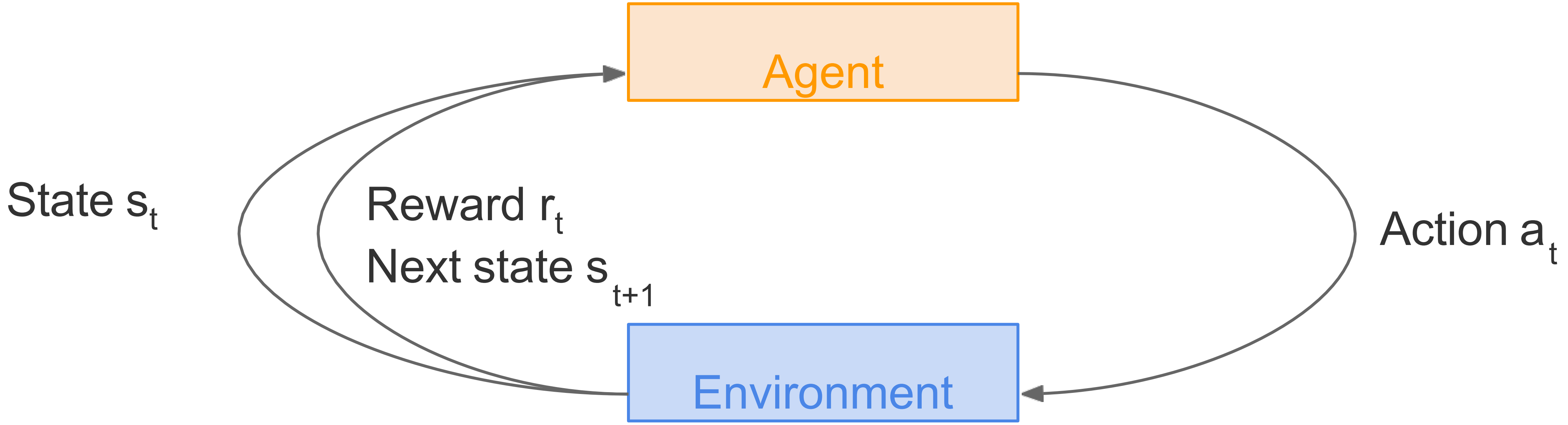
Outline

- Reinforcement learning for text generation

- 2 Paper presentations (15 x 2 mins)
  - **Ruisi Zhang:** Plug and Play Language Models: A Simple Approach to Controlled Text Generation
  - **Han Cao**: ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators

Reinforcement Learning

Recap: Reinforcement Learning



Goal: Learn how to take actions in order to maximize reward

Recap: Markov Decision Process

- At time step t=0, environment samples initial state $s_0 \sim p(s_0)$
- Then, for t=0 until done:
    - Agent selects action a_t
    - Environment samples reward $r_t \sim R(. \mid s_t, a_t)$
    - Environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
    - Agent receives reward r_t and next state s_{t+1}

- A policy π is a function from S to A that specifies what action to take in each state
- **Objective**: find policy π^* that maximizes cumulative discounted reward: $\sum_{t \geq 0} \gamma^t r_t$

Recap: Value function and Q-value function

Following a policy produces sample trajectories (or paths) s_0, a_0, r_0, s_1, a_1, r_1, …

How good is a state?
The **value function** at state s, is the expected cumulative reward from following the policy from state s:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, \pi\right]$$

How good is a state-action pair?
The **Q-value function** at state s and action a, is the expected cumulative reward from taking action a in state s and then following the policy:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Recap: Bellman equation

The optimal Q-value function Q* is the maximum expected cumulative reward achievable from a given (state, action) pair:

$$Q^*(s, a) = \max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

Q* satisfies the following **Bellman equation**:

$$Q^*(s, a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s', a') | s, a\right]$$

Recap: Q-learning

Remember: want to find a Q-function that satisfies the Bellman Equation:

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q^*(s', a') | s, a \right]$$

Q-learning: Use a function approximator to estimate the action-value function

$$Q(s,a;\theta) \approx Q^*(s,a)$$

Loss function: $L_i(\theta_i) = \mathbb{E}_{s,a \sim \rho(\cdot)} \left[(y_i - Q(s,a;\theta_i))^2 \right]$

where $y_i = \mathbb{E}_{s' \sim \mathcal{E}} \left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) | s, a \right]$

Recap: REINFORCE algorithm

Mathematically, we can write:

$$J(\theta) = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\right] = \int_\tau r(\tau)p(\tau;\theta)\mathrm{d}\tau$$

Where r(τ) is the reward of a trajectory $\tau = (s_0, a_0, r_0, s_1, \dots)$

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau)\nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta)\mathrm{d}\tau = \mathbb{E}_{\tau \sim p(\tau;\theta)}\left[r(\tau)\nabla_\theta \log p(\tau;\theta)\right]$$

When sampling a trajectory τ, we can estimate J(θ) with

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(τ) is high, push up the probabilities of the actions seen
- If r(τ) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

Intuition

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Interpretation:

- If r(r) is high, push up the probabilities of the actions seen
- If r(r) is low, push down the probabilities of the actions seen

Might seem simplistic to say that if a trajectory is good then all its actions were good. But in expectation, it averages out!

However, this also suffers from high variance because credit assignment is really hard. Can we help the estimator?

Variance reduction

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} r(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction

Gradient estimator: $$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} r(\tau) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

First idea: Push up probabilities of an action seen, only by the cumulative future reward from that state

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Second idea: Use discount factor γ to ignore delayed effects

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

Variance reduction: Baseline

Problem: The raw value of a trajectory isn't necessarily meaningful. For example, if rewards are all positive, you keep pushing up probabilities of actions.

What is important then? Whether a reward is better or worse than what you expect to get

Idea: Introduce a baseline function dependent on the state.
Concretely, estimator is now:

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t\geq 0} \left(\sum_{t'\geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

How to choose the baseline?

$$\nabla_\theta J(\theta) \approx \sum_{t \geq 0} \left(\sum_{t' \geq t} \gamma^{t'-t} r_{t'} - b(s_t) \right) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

A simple baseline: constant moving average of rewards experienced so far from all trajectories

Variance reduction techniques seen so far are typically used in “Vanilla REINFORCE”

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action a_t in a state s_t if $Q^\pi(s_t, a_t) - V^\pi(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

How to choose the baseline?

A better baseline: Want to push up the probability of an action from a state, if this action was better than the **expected value of what we should get from that state**.

Q: What does this remind you of?

A: Q-function and value function!

Intuitively, we are happy with an action a_t in a state s_t if $Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$ is large. On the contrary, we are unhappy with an action if it's small.

Using this, we get the estimator: $\nabla_\theta J(\theta) \approx \sum_{t \geq 0} (Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)) \nabla_\theta \log \pi_\theta(a_t | s_t)$

Actor-Critic Algorithm

Problem: we don't know Q and V. Can we learn them?

Yes, using Q-learning! We can combine Policy Gradients and Q-learning by training both an **actor** (the policy) and a **critic** (the Q-function).

- The actor decides which action to take, and the critic tells the actor how good its action was and how it should adjust
- Also alleviates the task of the critic as it only has to learn the values of (state, action) pairs generated by the policy
- Can also incorporate Q-learning tricks e.g. experience replay
- **Remark:** we can define by the **advantage function** how much an action was better than expected

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ø

For iteration=1, 2 … **do**

Sample m trajectories under the current policy

$\Delta\theta \leftarrow 0$

For i=1, …, m **do**

For t=1, ... , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

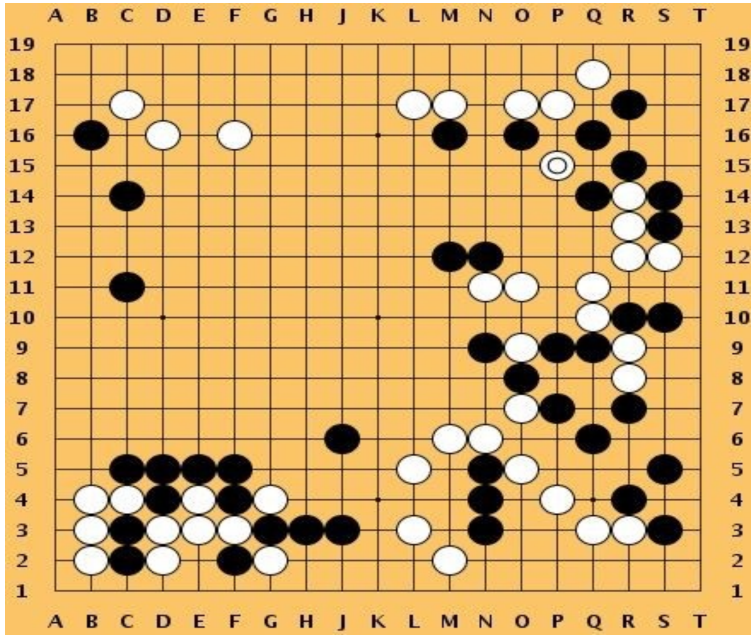
$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$$\theta \leftarrow \alpha\Delta\theta$$

$$\phi \leftarrow \beta\Delta\phi$$

End for

More policy gradients: AlphaGo



Overview:

- Mix of supervised learning and reinforcement learning
- Mix of old methods (Monte Carlo Tree Search) and recent ones (deep RL)

How to beat the Go world champion:

- Featurize the board (stone color, move legality, bias, …)
- Initialize policy network with supervised training from professional go games, then continue training using policy gradient (play against itself from random previous iterations, +1 / -1 reward for winning / losing)
- Also learn value network (critic)
- Finally, combine combine policy and value networks in a Monte Carlo Tree Search algorithm to select actions by lookahead search

[Silver et al., Nature 2016]

RL for Text Generation

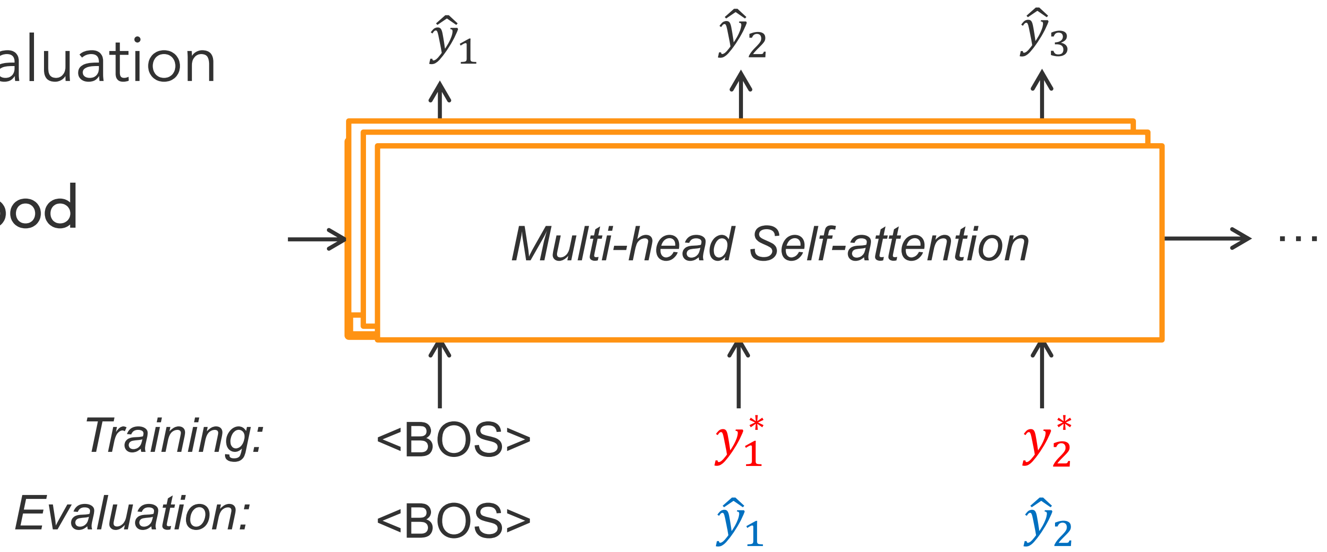
Two Central Goals

- Generating human-like, grammatical, and readable text
  - I.e., generating **natural** language
- Generating text that contains desired information inferred from inputs
  - Machine translation
    - Source sentence --> target sentence w/ the same meaning
  - Data description
    - Table --> data report describing the table
  - Attribute control
    - Sentiment: positive --> ``I like this restaurant''
  - Conversation control
    - Control conversation strategy and topic

Two Issues of MLE

- Exposure bias [Ranzato et al., 2015]
  - **Training**: predict next token given the previous ground-truth sequence
  - **Evaluation**: predict next token given the previous sequence that are generated by the model itself

- Mismatch between training & evaluation criteria
  - Train to maximize **data log-likelihood**
  - Evaluate with, e.g., **BLEU**

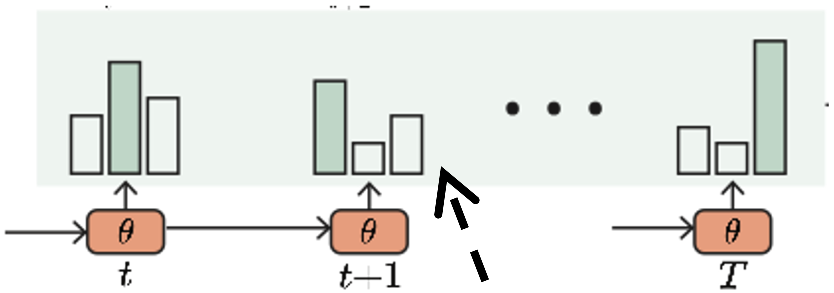


[Ranzato et al., 2015] Sequence Level Training with Recurrent Neural Networks

Reinforcement Learning (RL)

- Plug in arbitrary reward functions to drive learning
- Fertile research area for robotic and game control
- But … limited success for training text generation
  - Challenges:
    - Large sequence space: (vocab-size)$^{\text{text-length}}$ ~ $(10^6)^{20}$
    - Sparse reward: only after seeing the whole text sequence
  - Impossible to train from scratch, usually initialized with MLE
  - Unclear improvement vs MLE

RL for Text Generation



- (Autoregressive) text generation model:

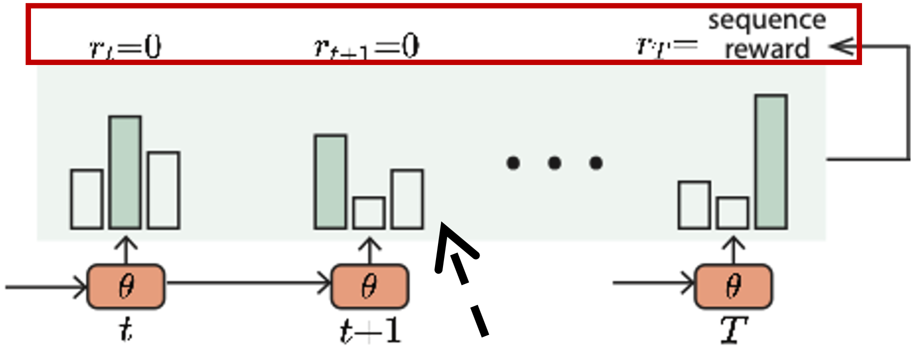
Sentence $\boldsymbol{y} = (y_0, \ldots, y_T)$

$$\pi_\theta(y_t \mid \boldsymbol{y}_{<t}) = \frac{\exp f_\theta(y_t | \boldsymbol{y}_{<t})}{\sum_{y'} \exp f_\theta(y' | \boldsymbol{y}_{<t})}$$

logits

In RL terms: trajectory, τ action, a_t state, $\boldsymbol{s}_t$ policy $\pi_\theta(a_t \mid \boldsymbol{s}_t)$

RL for Text Generation



- (Autoregressive) text generation model:

Sentence $\boldsymbol{y} = (y_0, \dots, y_T)$

$$\pi_\theta(y_t \mid \boldsymbol{y}_{<t}) = \frac{\exp f_\theta(y_t|\boldsymbol{y}_{<t})}{\sum_{y'} \exp f_\theta(y'|\boldsymbol{y}_{<t})}$$

logits

In RL terms: trajectory, τ action, a_t state, $\boldsymbol{s}_t$ policy $\pi_\theta(a_t \mid \boldsymbol{s}_t)$

- Reward $r_t = r(\boldsymbol{s}_t, a_t)$
  - Often **sparse**: $r_t = 0$ for $t < T$

RL for Text Generation: REINFORCE

Given a dataset of input output pairs, $\mathcal{D} \equiv \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)*})\}_{i=1}^{N}$

learn a conditional distribution $p_\theta(\mathbf{y} \mid \mathbf{x})$ that minimizes

expected loss:

$$\mathcal{L}_{\mathrm{RL}}(\boldsymbol{\theta}) = \sum_{(\mathbf{x}, \mathbf{y}^*) \in \mathcal{D}} - \sum_{\mathbf{y} \in \mathcal{Y}} p_\theta(\mathbf{y} \mid x)\; r(\mathbf{y}, \mathbf{y}^*)$$

Sample from the ***model*** *distribution*

Slide courtesy: Russ Salakhutdinov @ CMU 10707

RL for Text Generation: REINFORCE

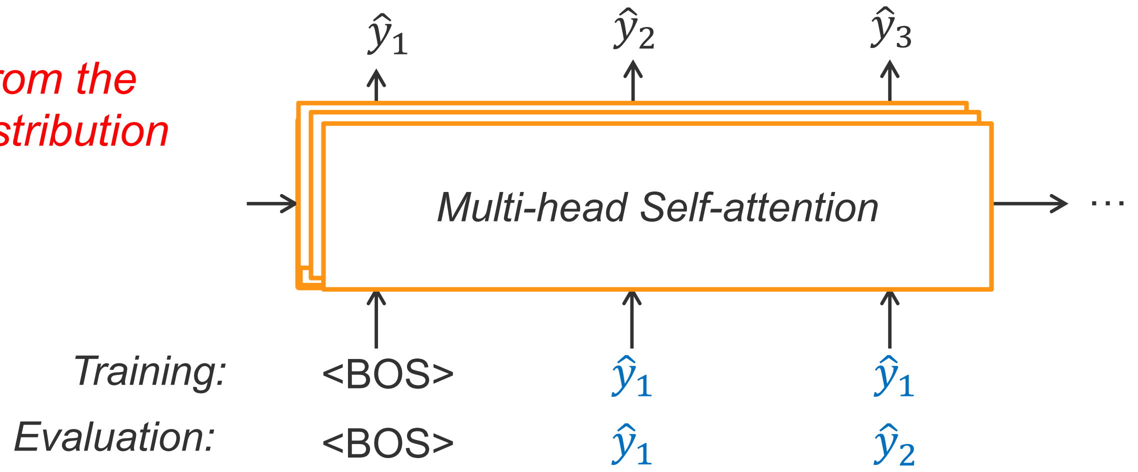
Given a dataset of input output pairs, $\mathcal{D} \equiv \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)*})\}_{i=1}^{N}$

learn a conditional distribution $p_\theta(\mathbf{y} \mid \mathbf{x})$ that minimizes

expected loss:

$$\mathcal{L}_{\mathrm{RL}}(\boldsymbol{\theta}) = \sum_{(\mathbf{x},\mathbf{y}^*)\in\mathcal{D}} -\sum_{\mathbf{y}\in\mathcal{Y}} p_\theta(\mathbf{y} \mid x)\; r(\mathbf{y},\mathbf{y}^*)$$

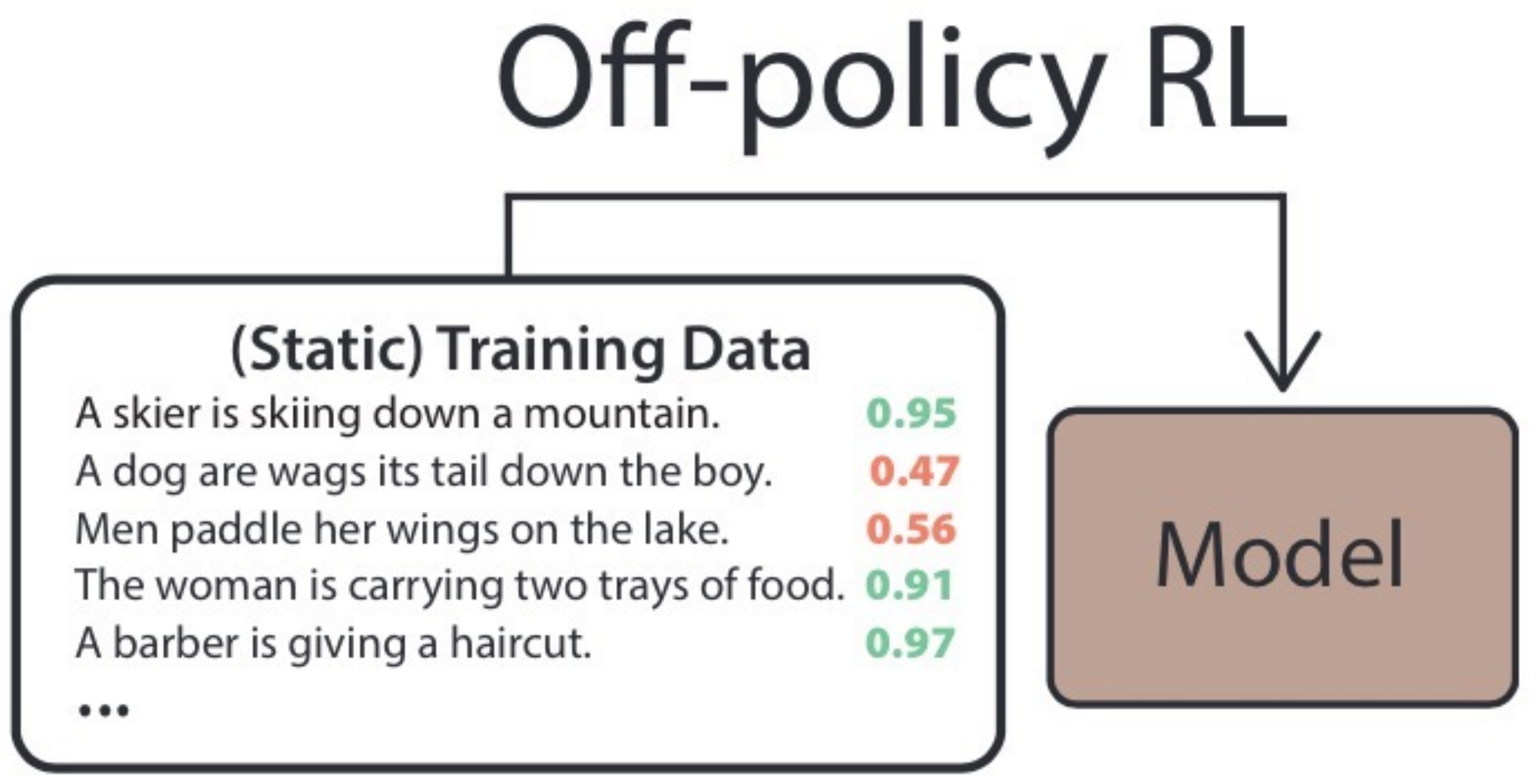
Sample from the ***model*** *distribution*



No exposure bias

Slide courtesy: Russ Salakhutdinov @ CMU 10707

RL for Text Generation



- Off-policy RL
  - *e.g., Q-learning*
  - Implicitly learns the policy π by approximating the $Q^{\pi}(\boldsymbol{s}_t, a_t)$
  - Bellman temporal consistency: $Q^*(\boldsymbol{s}_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q^*(\boldsymbol{s}_{t+1}, a_{t+1})$
  - Learns Q_θ with the regression objective:

target Q-network

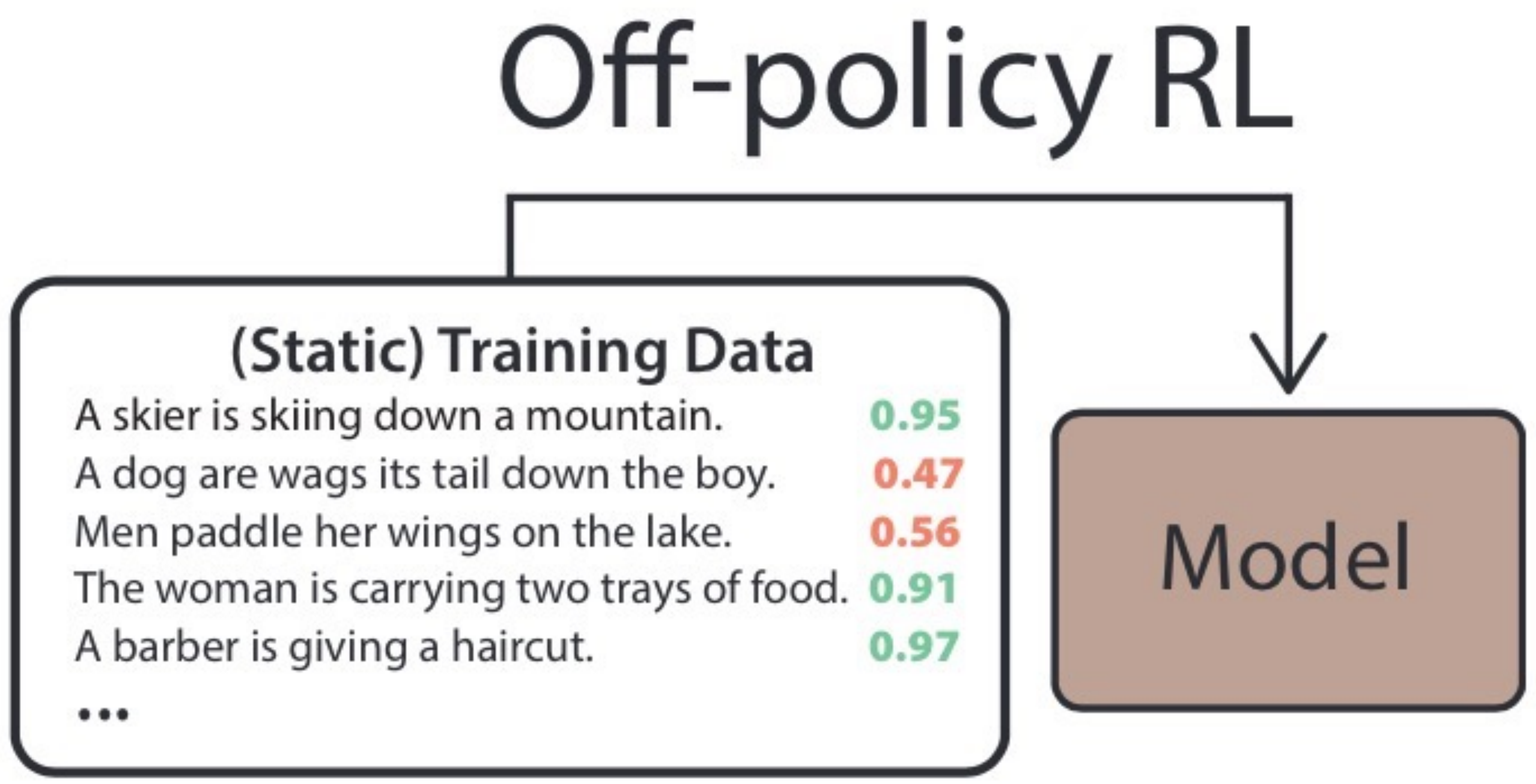
$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(\boxed{r_t + \gamma \max_{a_{t+1}} Q_{\bar{\theta}}(\boldsymbol{s}_{t+1}, a_{t+1})} - Q_\theta(\boldsymbol{s}_t, a_t)\right)^2\right]$$

Regression target

Arbitrary policy, e.g., training data

  - After learning, induces the policy as $a_t = \operatorname{argmax}_a Q_{\theta^*}(\boldsymbol{s}_t, a)$

RL for Text Generation



- Off-policy RL
  - *e.g., Q-learning*
  - Implicitly learns the policy π by approximating the $Q^{\pi}(\boldsymbol{s}_t, a_t)$
  - Bellman temporal consistency: $Q^*(\boldsymbol{s}_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q^*(\boldsymbol{s}_{t+1}, a_{t+1})$
  - Learns Q_θ with the regression objective:

Slow updates: gradient involves only Q_θ-value of **one** action a_t (vs 10^6 vocab size)

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(\boxed{r_t + \gamma \max_{a_{t+1}} Q_{\bar{\theta}}(\boldsymbol{s}_{t+1}, a_{t+1})} - Q_\theta(\boldsymbol{s}_t, a_t)\right)^2\right]$$

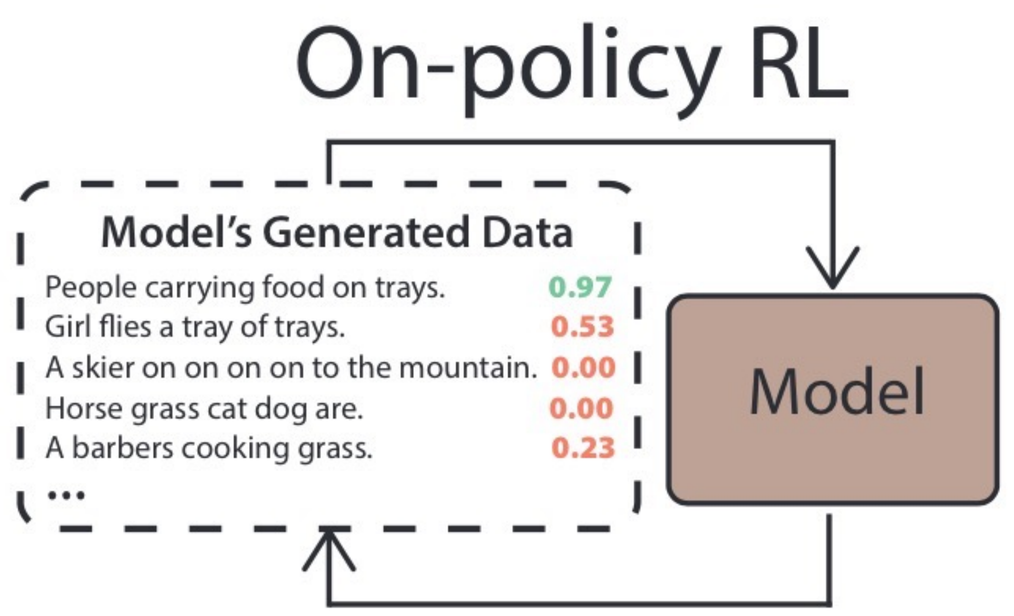
Arbitrary policy, e.g., training data

Regression target is unstable
- Bootstrapped $Q_{\bar{\theta}}$
- Sparse reward $r_t = 0$ $(t < T)$: no "true" training signal

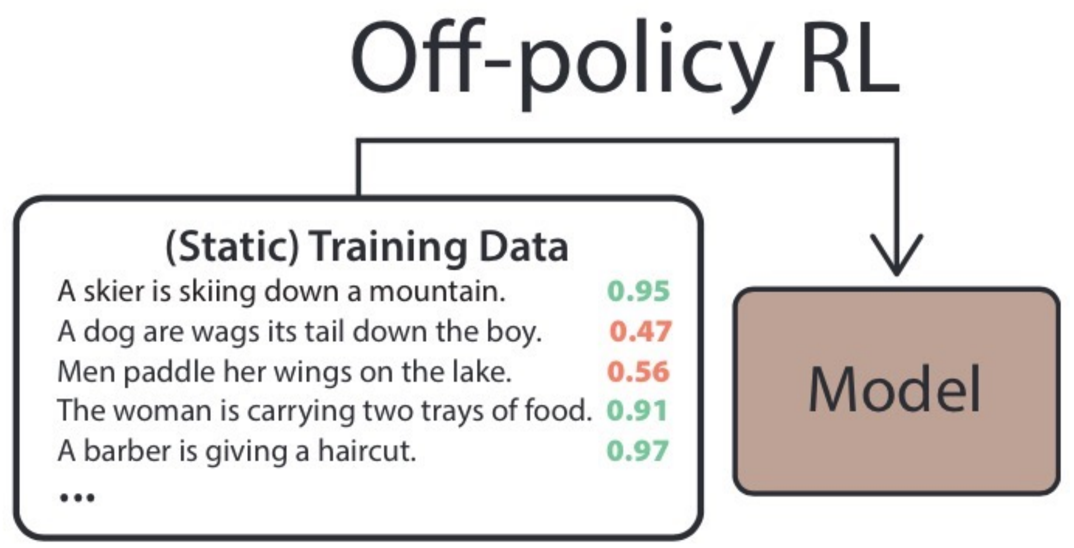
  - After learning, induces the policy as $a_t = \text{argmax}_a\, Q_{\theta^*}(\boldsymbol{s}_t, a)$

RL for Text Generation

- On-policy RL, *e.g., Policy Gradient (PG)*
  - Exploration to maximize reward directly
  - Extremely low data efficiency



- Off-policy RL, *e.g., Q-learning*
  - Unstable training due to bootstrapping & sparse reward
  - Slow updates due to large action space
  - Sensitive to training data quality; lacks on-policy exploration



New RL for Text Generation: Soft Q-Learning (SQL)

(Hard) Q-learning

- Goal

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$$

- Induced policy

$$a_t = \operatorname{argmax}_a Q_{\theta^*}(\boldsymbol{s}_t, a)$$

SQL

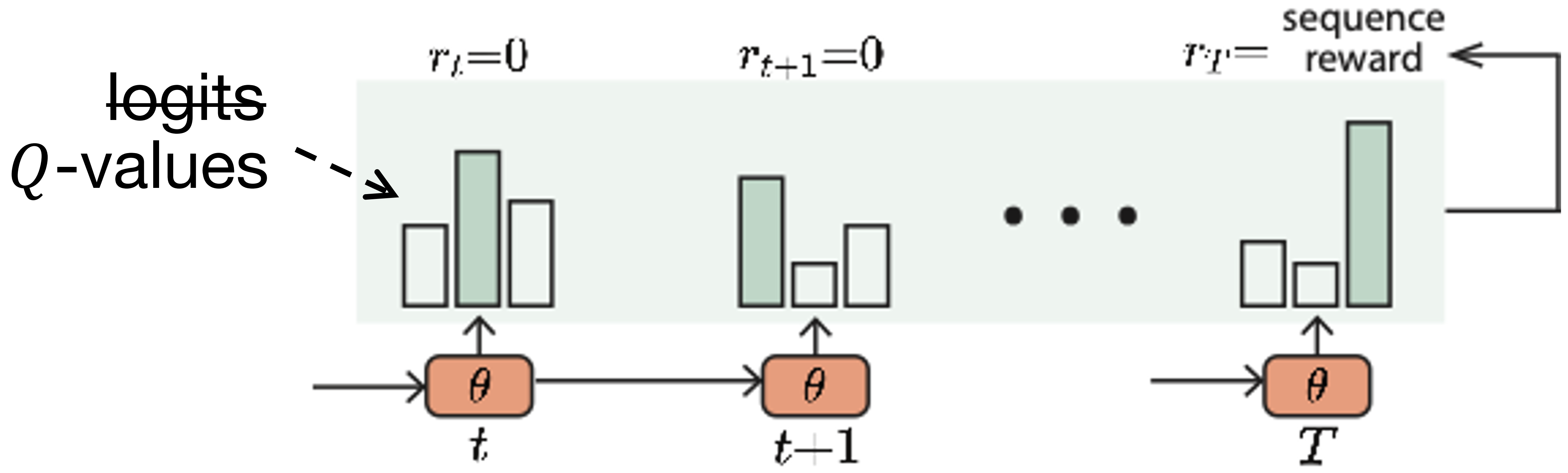
- Goal: entropy regularized

$$J_{\text{MaxEnt}}(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{T} \gamma^t r_t + \alpha \mathcal{H}(\pi(\cdot \mid \boldsymbol{s}_t))\right]$$

- Induced policy

$$\pi_{\theta^*}(a_t \mid \boldsymbol{s}_t) = \frac{\exp Q_{\theta^*}(a_t|\boldsymbol{s}_t)}{\sum_a \exp Q_{\theta^*}(a|\boldsymbol{s}_t)}$$

Generation model's "logits" now act as Q-values !



New RL for Text Generation: Soft Q-Learning (SQL)

(Hard) Q-learning

- Goal

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t\right]$$

- Induced policy

$$a_t = \mathrm{argmax}_a\, Q_{\theta^*}(\boldsymbol{s}_t, a)$$

- Training objective:
  - Based on temporal consistency

😈 Unstable training / slow updates

SQL

- Goal: entropy regularized

$$J_{\mathrm{MaxEnt}}(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^t r_t + \alpha\mathcal{H}\left(\pi\left(\cdot \mid \boldsymbol{s}_t\right)\right)\right]$$

- Induced policy

$$\pi_{\theta^*}(a_t \mid \boldsymbol{s}_t) = \frac{\exp Q_{\theta^*}(a_t|\boldsymbol{s}_t)}{\sum_a \exp Q_{\theta^*}(a|\boldsymbol{s}_t)}$$

- Training objective:
  - Based on **path consistency**

😇 Stable / efficient

Efficient Training via Path Consistency

$$V^*(\boldsymbol{s}) = \log \sum_{a'} \exp Q^*(\boldsymbol{s}, a')$$

$$\pi^*(a \mid \boldsymbol{s}) = \frac{\exp Q^*(\boldsymbol{s}, a)}{\sum_{a'} \exp Q^*(\boldsymbol{s}, a')}$$

- (Single-step) path consistency

$$V^*(\boldsymbol{s}_t) - \gamma V^*(\boldsymbol{s}_{t+1}) = r_t - \log \pi^*(a_t \mid \boldsymbol{s}_t)$$

  - Objective

Regression target

$$\mathcal{L}_{\mathrm{SQL,\,PCL}}(\boldsymbol{\theta}) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(\boxed{-V_{\bar{\theta}}(\boldsymbol{s}_t) + \gamma V_{\bar{\theta}}(\boldsymbol{s}_{t+1}) + r_t} - \log \pi_\theta(a_t \mid \boldsymbol{s}_t)\right)^2\right]$$

$\approx A_{\bar{\theta}}(\boldsymbol{s}_t, a_t)$, advantage

😇 Fast updates: gradient involves Q_θ values of **all** tokens in the vocab

SQL matches log probability of token a_t with its advantage

v.s.

MLE increases log probability of token a_t blindly

[Nachum et al., 2017]

Efficient Training via Path Consistency

$$V^*(\boldsymbol{s}) = \log \sum_{a'} \exp Q^*(\boldsymbol{s}, a')$$

$$\pi^*(a \mid \boldsymbol{s}) = \frac{\exp Q^*(\boldsymbol{s}, a)}{\sum_{a'} \exp Q^*(\boldsymbol{s}, a')}$$

- (Single-step) path consistency

$$V^*(\boldsymbol{s}_t) - \gamma V^*(\boldsymbol{s}_{t+1}) = r_t - \log \pi^*(a_t \mid \boldsymbol{s}_t)$$

  - Objective

Regression target

$$\mathcal{L}_{\text{SQL, PCL}}(\boldsymbol{\theta}) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(\boxed{-V_{\bar{\theta}}(\boldsymbol{s}_t) + \gamma V_{\bar{\theta}}(\boldsymbol{s}_{t+1}) + r_t} - \log \pi_\theta(a_t \mid \boldsymbol{s}_t)\right)^2\right]$$

Fast updates: gradient involves Q_θ values of *all* tokens in the vocab

- (Multi-step) path consistency

$$V^*(\boldsymbol{s}_t) - \gamma^{T-t} V^*(\boldsymbol{s}_{T+1}) = \sum_{l=0}^{T-t} \gamma^l \left(r_{t+l} - \log \pi^*(a_{t+l} \mid \boldsymbol{s}_{t+l})\right)$$

  - Objective

$$\mathcal{L}_{\text{SQL, PCL-ms}}(\boldsymbol{\theta}) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(\boxed{-V_{\bar{\theta}}(\boldsymbol{s}_t) + \gamma^{T-t} r_T} - \sum_{l=0}^{T-t} \gamma^l \log \pi_\theta(a_{t+l} \mid \boldsymbol{s}_{t+l})\right)^2\right]$$

Stable updates: Non-zero reward signal r_T as regression target

[Nachum et al., 2017]

Efficient Training via Path Consistency

$$V^*(\boldsymbol{s}) = \log \sum_{a'} \exp Q^*(\boldsymbol{s}, a')$$

$$\pi^*(a \mid \boldsymbol{s}) = \frac{\exp Q^*(\boldsymbol{s}, a)}{\sum_{a'} \exp Q^*(\boldsymbol{s}, a')}$$

- (Single-step) path consistency

$$V^*(\boldsymbol{s}_t) - \gamma V^*(\boldsymbol{s}_{t+1}) = r_t - \log \pi^*(a_t \mid \boldsymbol{s}_t)$$

- Objective

Regression target

$$\mathcal{L}_{\text{SQL, PCL}}(\boldsymbol{\theta}) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(\boxed{-V_{\bar{\theta}}(\boldsymbol{s}_t) + \gamma V_{\bar{\theta}}(\boldsymbol{s}_{t+1}) + r_t} - \log \pi_\theta(a_t \mid \boldsymbol{s}_t)\right)^2\right]$$

Fast updates: gradient involves Q_θ values of **all** tokens in the vocab

Arbitrary policy:
- Training data (if available) → off-policy updates
- Current policy → on-policy updates
- We combine both for the best of the two

Stable updates: Non-zero reward signal r_T as regression target

$$\mathcal{L}_{\text{SQL, PCL-ms}}(\boldsymbol{\theta}) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(\boxed{-V_{\bar{\theta}}(\boldsymbol{s}_t) + \gamma^{T-t} r_T} - \sum_{l=0}^{T-t} \gamma^l \log \pi_\theta(a_{t+l} \mid \boldsymbol{s}_{t+l})\right)^2\right]$$

Implementation is easy

```
model = TransformerLM(...)

for iter in range(max_iters):
    if mode == "off-policy":
        batch = dataset.sample_batch()
        sample_ids = batch.text_ids

    if mode == "on-policy":
        sample_ids = model.decode()

    Q_values = model.forward(sample_ids)
    Q_values_target = target_model.forward(sample_ids)

    rewards = compute_rewards(sample_ids)

    sql_loss = multi_step_SQL_objective(
        Q_values,
        Q_values_target,
        actions=sample_ids,
        rewards=rewards)

    # gradient descent over sql_loss
    # ...
```

```
def multi_step_SQL_objective(
        Q_values, Q_values_target, actions, rewards):

    V = Q_values.logsumexp(dim=-1)
    A = Q_values[actions] - V

    V_target = Q_values_target.logsumexp(dim=-1)

    A2 = masked_reverse_cumsum(
        A, lengths=actions.sequence_length,
        dim=-1)

    return F.mse_loss(
        A2, rewards.view(-1, 1) - V_target,
        reduction="none")
```

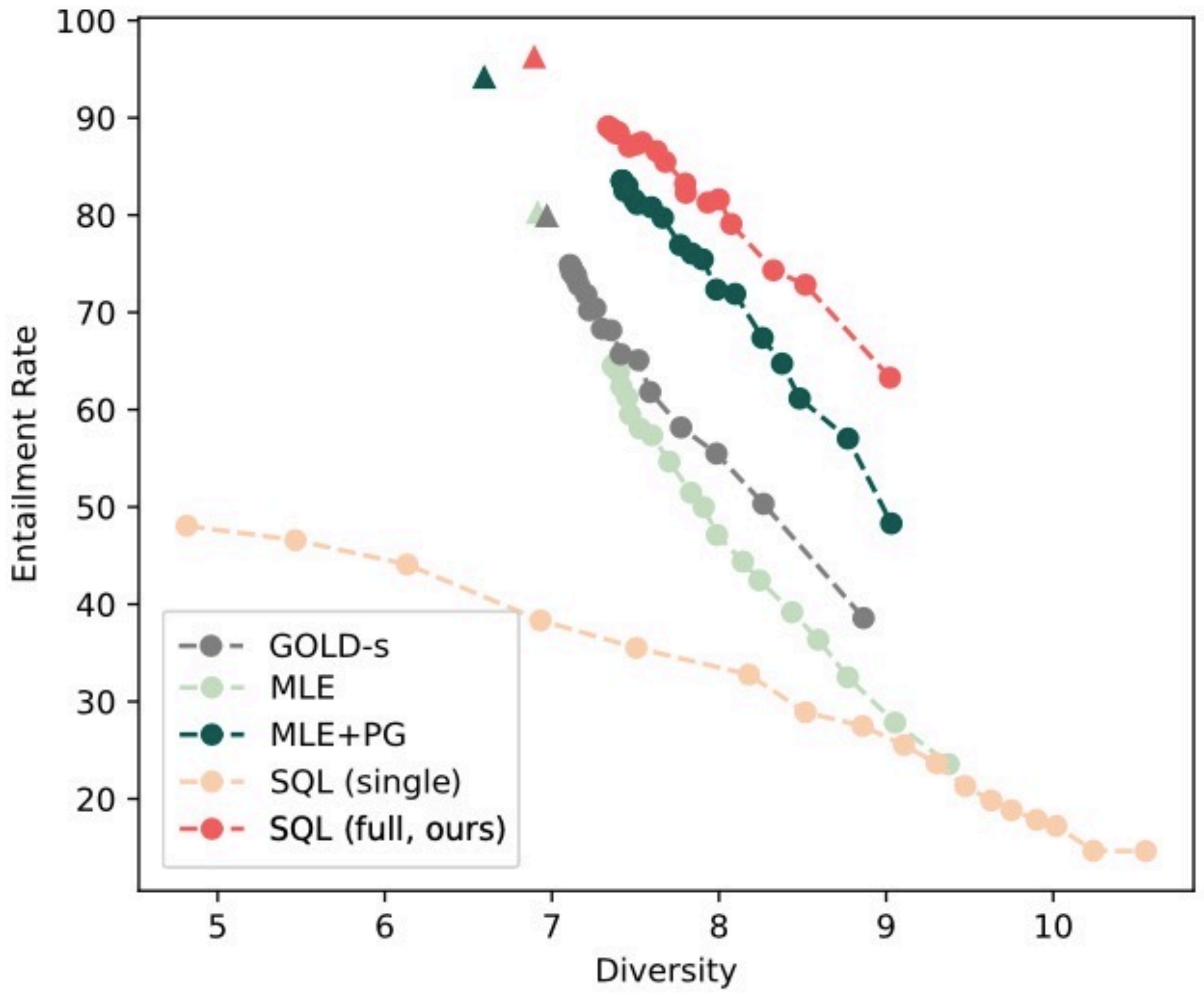
Applications & Experiments

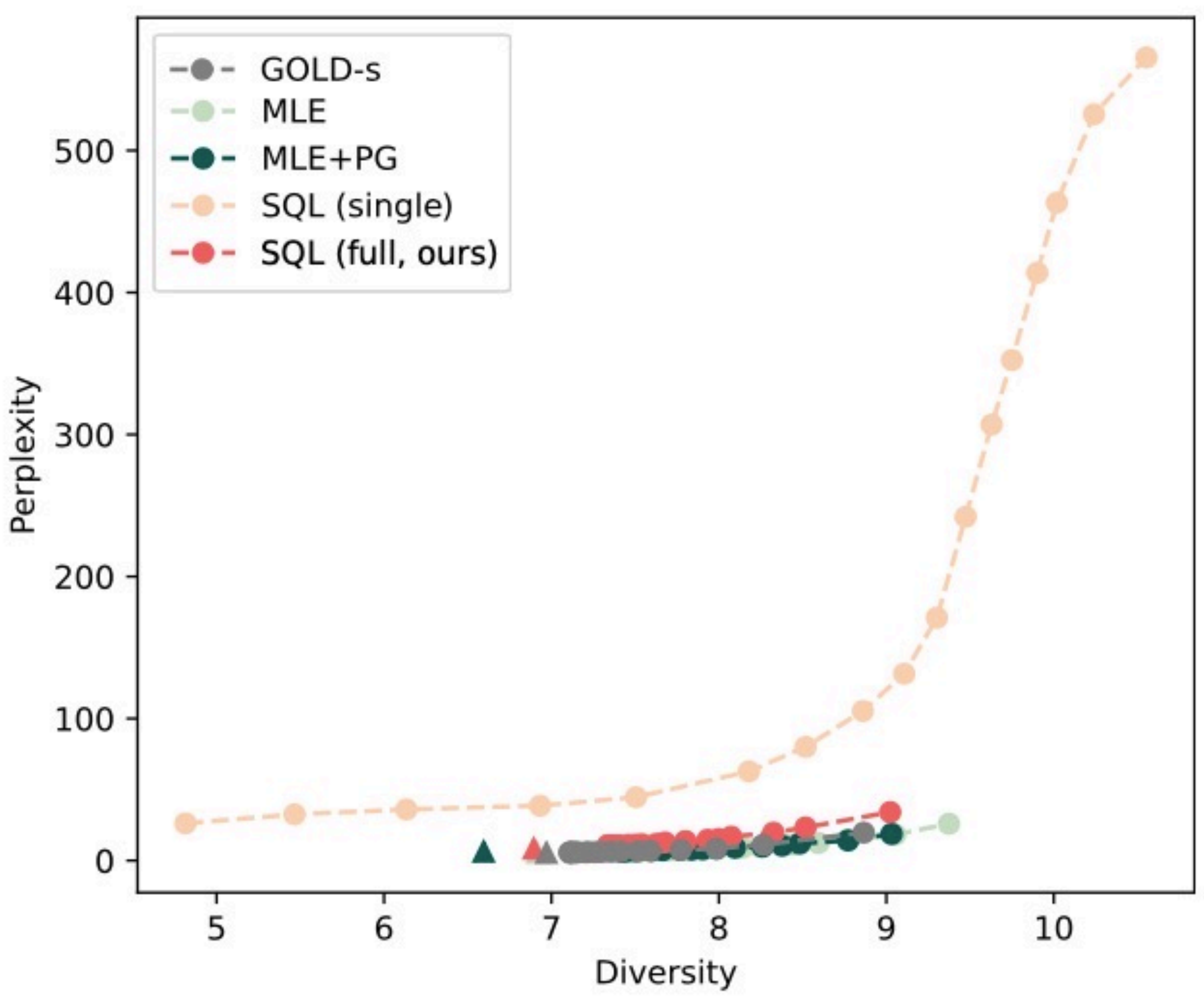
Application (I): Learning from Noisy (Negative) Text

- Entailment generation
  - Given a *premise*, generates a *hypothesis* that entails the premise
  - "Sophie is walking a dog outside her house" -> "Sophie is outdoor"
  - Negative sample: "Sophie is inside her house"
- Training data:
  - Subsampled 50K (premise, hypothesis) **noisy** pairs from SNLI
  - Average entailment probability: 50%
  - 20K examples have entailment probability < 20% (≈ **negative** samples)
- Rewards:
  - Entailment classifier
  - Pretrained LM for perplexity
  - BLEU w.r.t input premises (which effectively prevents trivial generations)

Application (I): Learning from Noisy (Negative) Text

- **MLE** and pure off-policy RL (**GOLD-s**) do not work ← rely heavy on data quality
- **SQL (full)** > **MLE+PG** (PG alone does not work)
- **SQL** (**single**-step only) does not work: the multi-step SQL objective is crucial

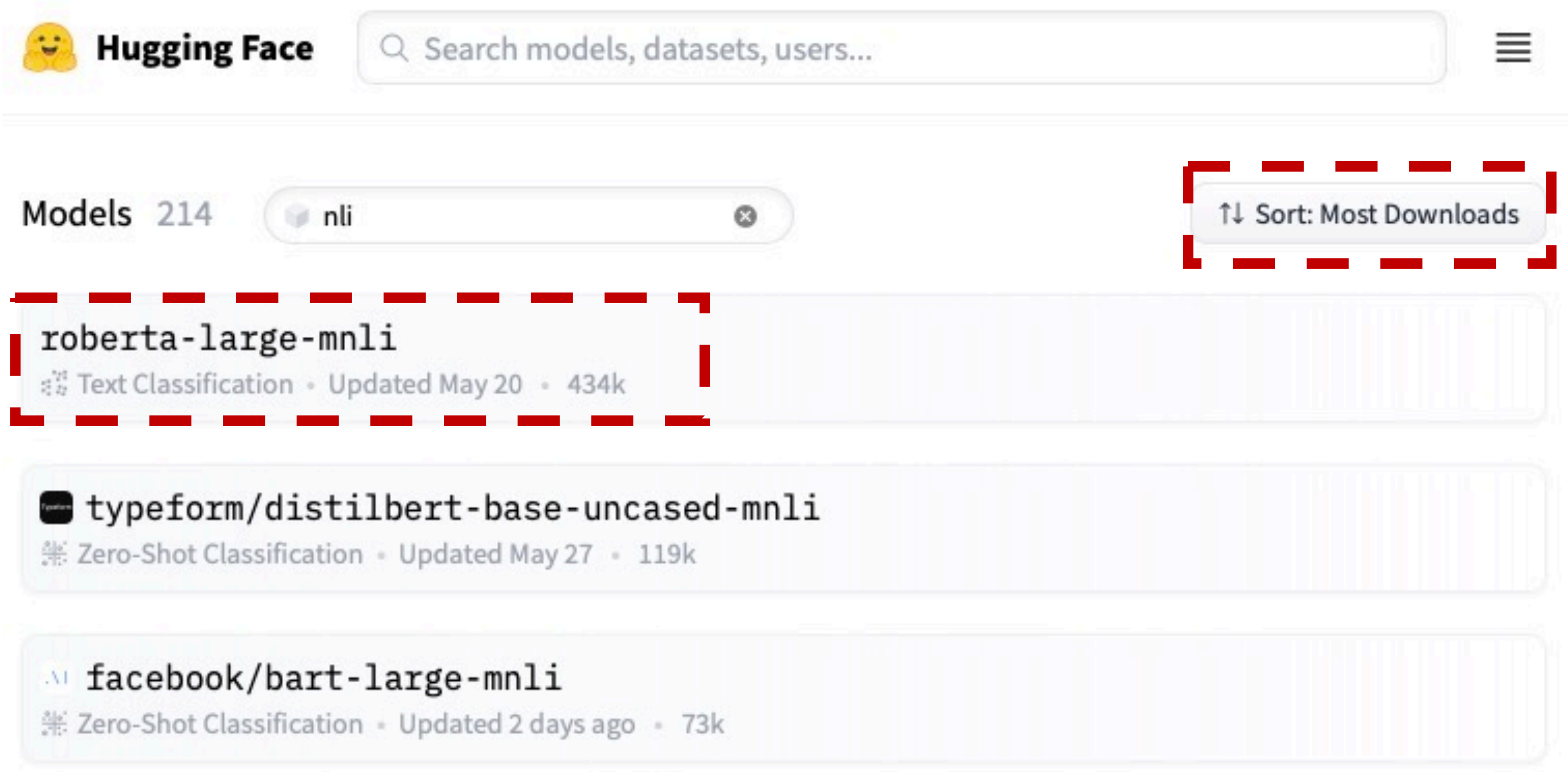




Entailment-rate and language-quality vs diversity (top-p decoding w/ different p)

Application (II): Universal Adversarial Attacks

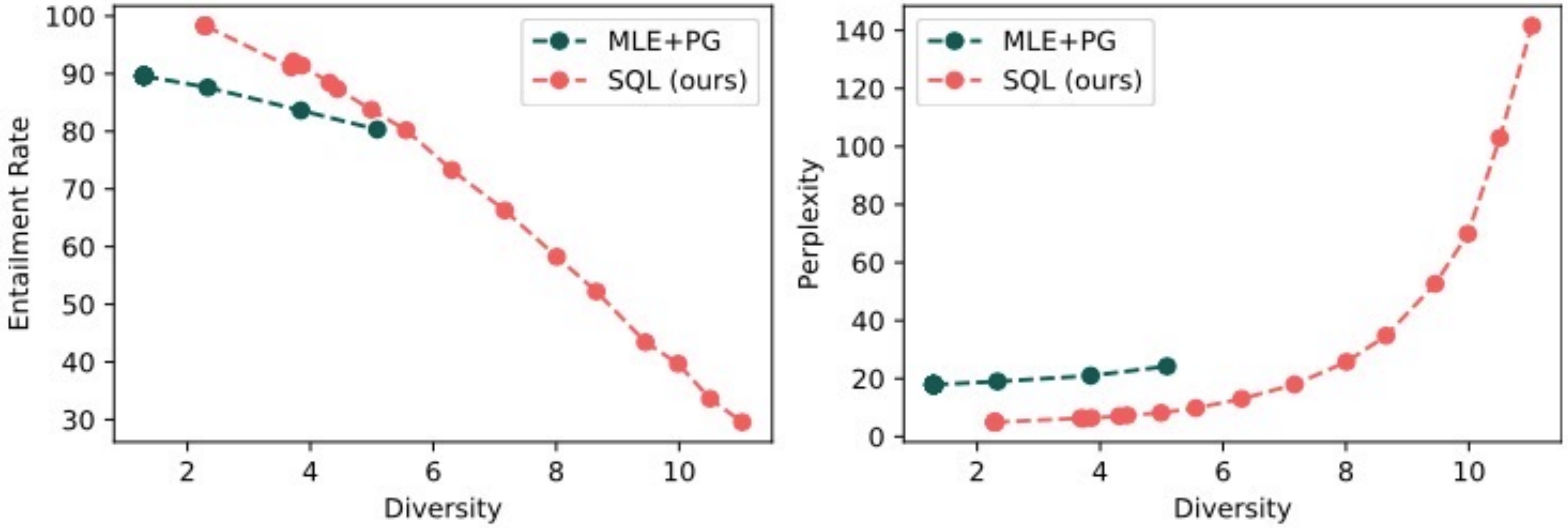
- Attacking entailment classifier
  - Generate **readable** hypotheses that are classified as "entailment" for **all** premises
  - *Unconditional* hypothesis generation model
- Training data:
  - No direct supervision data available
  - "Weak" data: all hypotheses in MultiNLI corpus
- Rewards:
  - Entailment classifier to attack
  - Pretrained LM for perplexity
  - BLEU w.r.t input premises
  - Repetition penalty



Previous adversarial algorithms are not applicable here:
- only attack for specific premise
- not readable

Application (II): Universal Adversarial Attacks

- **SQL (full)** > **MLE+PG** (PG alone does not work)
- **MLE+PG** collapses: cannot generate more diverse samples

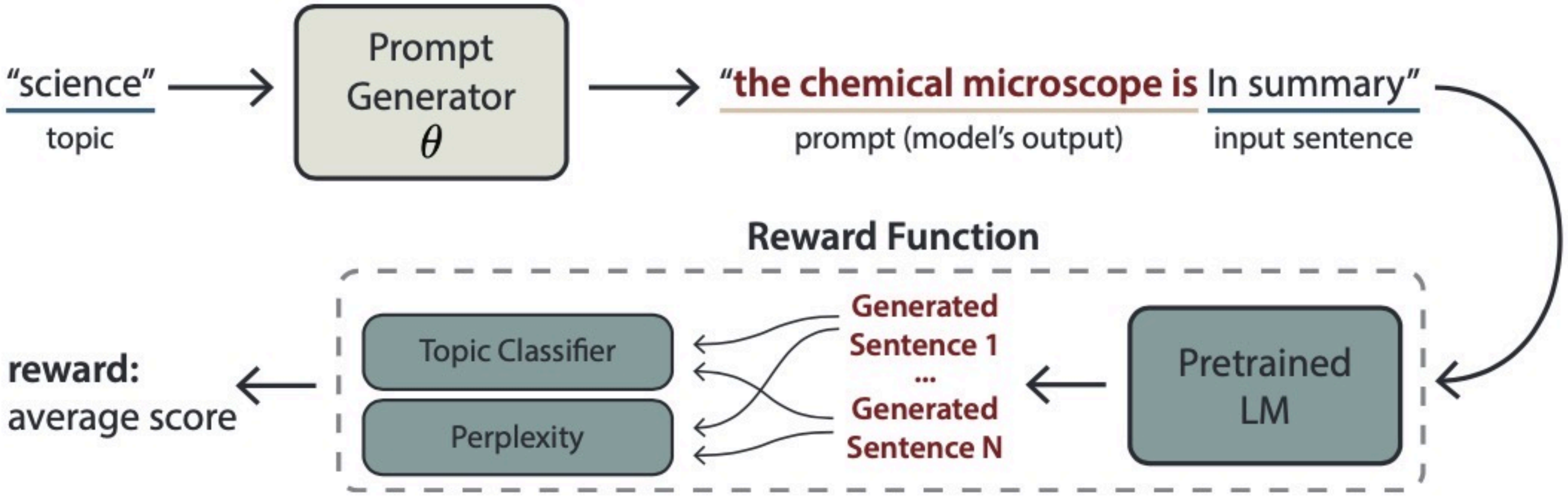


Model	Generation	Rate
MLE+PG	it 's .	90.48
SQL (ours)	the person saint-pierre-et-saint-paul is saint-pierre-et-saint-paul .	97.40

Samples of highest attack rate

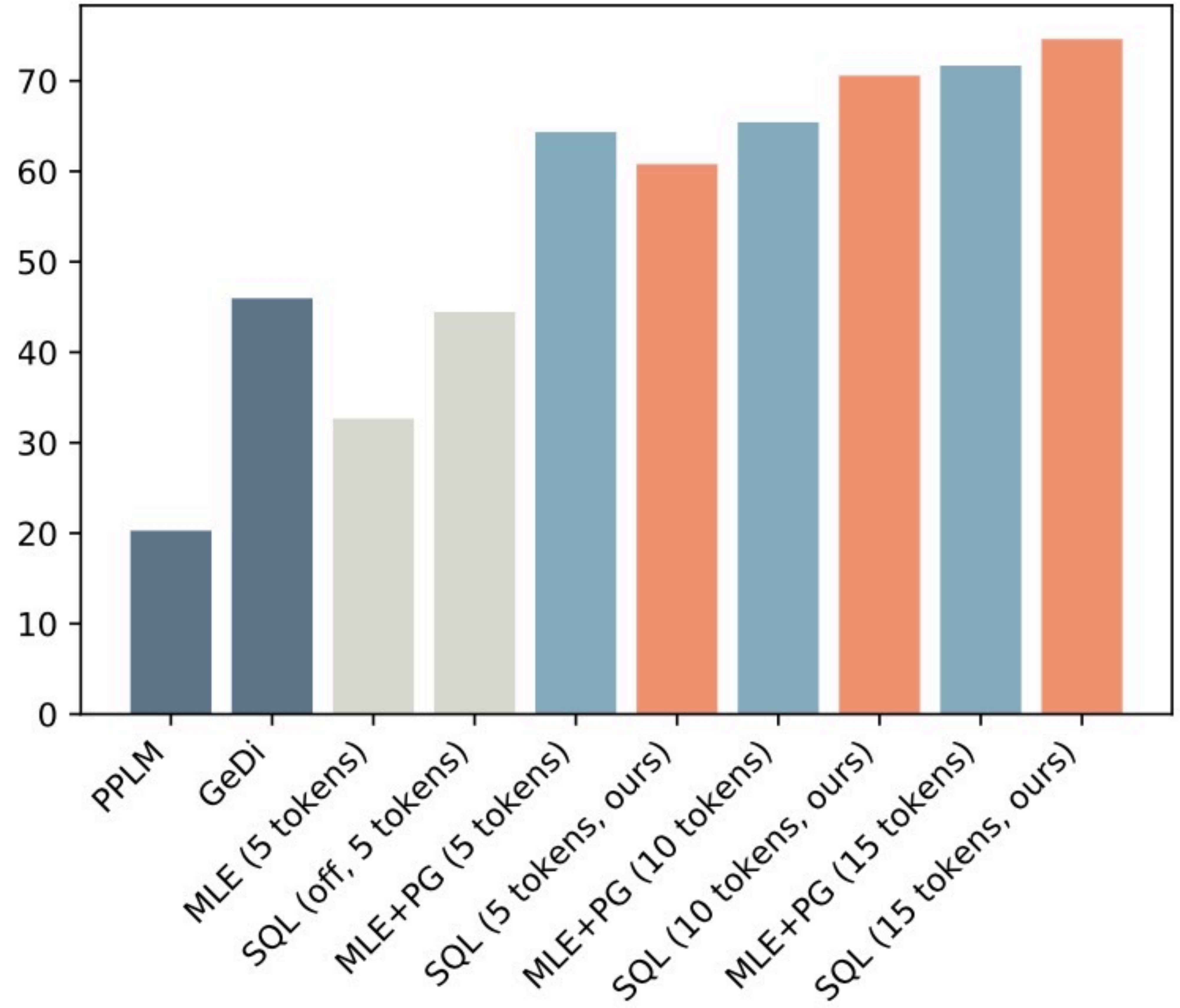
Application (III): Prompt Generation for Controlling LMs

- Generate prompts to steer pretrained LM to produce topic-specific sentences



Existing gradient-based prompt tuning methods are not applicable due to discrete components

Application (III): Prompt Generation for Controlling LMs



Topic accuracy

- Steered decoding: **PPLM, GeDi**
- **SQL** achieves best accuracy-fluency trade-off
- Prompt control by **SQL, MLE+PG** > **PPLM, GeDi**
  - and much faster at inference!
- **SQL (off-policy only)** > **MLE**

PPLM	GeDi	MLE (5)	SQL (off, 5)
12.69	123.88	25.70	25.77
MLE+PG (5/10/15)		**SQL (5/10/15, ours)**	
25.52/28.16/28.71		25.94/26.95/29.10	

Language perplexity

Model	PPLM	GeDi	SQL
Seconds	5.58	1.05	0.07

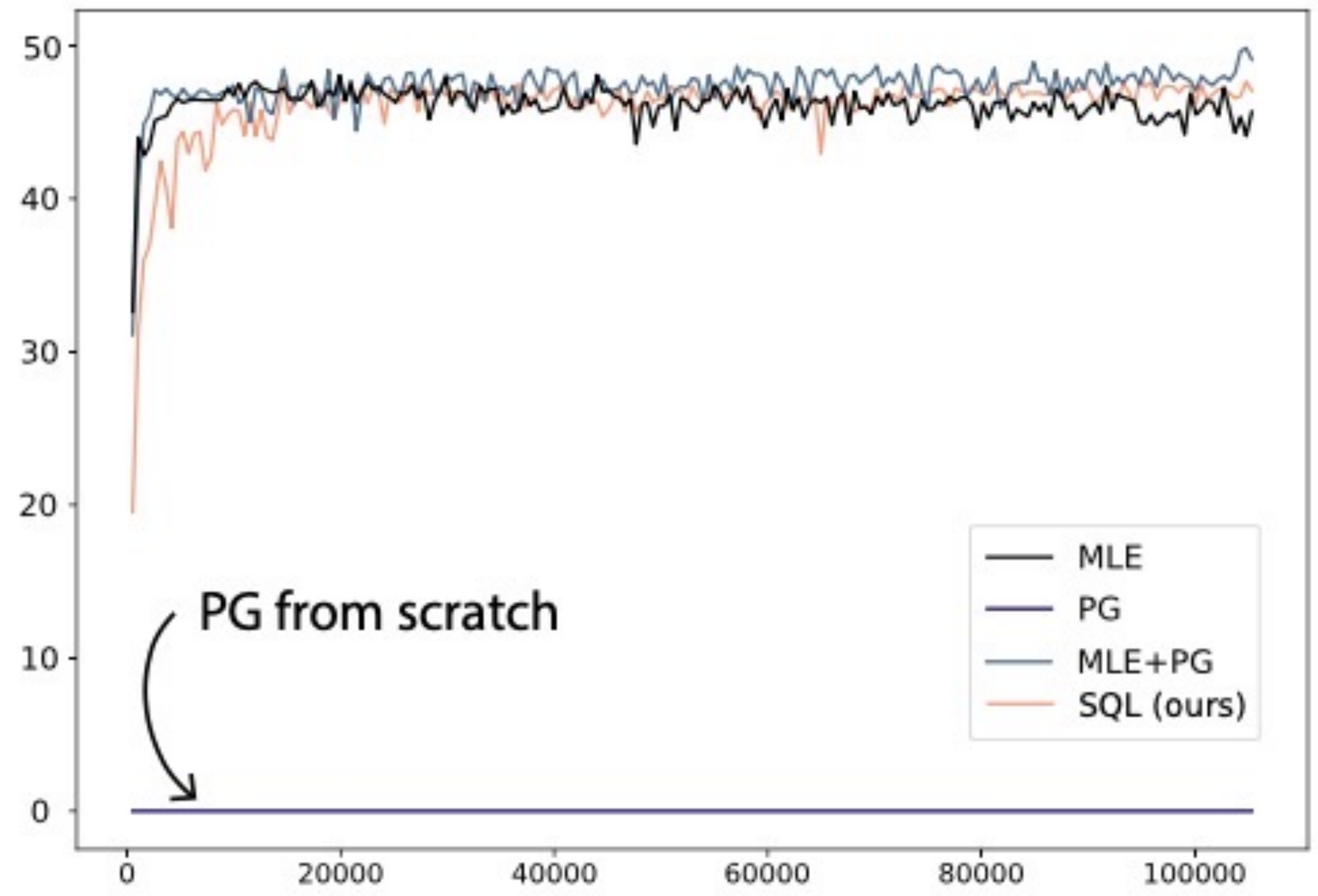
Time cost for generating one sentence

Promising results on standard supervised tasks

- **SQL** from scratch is competitive with **MLE** in terms of performance and stability
  - Results on E2E dataset
  - **PG** from scratch fails

Model	MLE	PG	MLE+PG	SQL (ours)
val	45.67	0.00	49.08	47.04
test	41.75	0.00	42.26	41.70

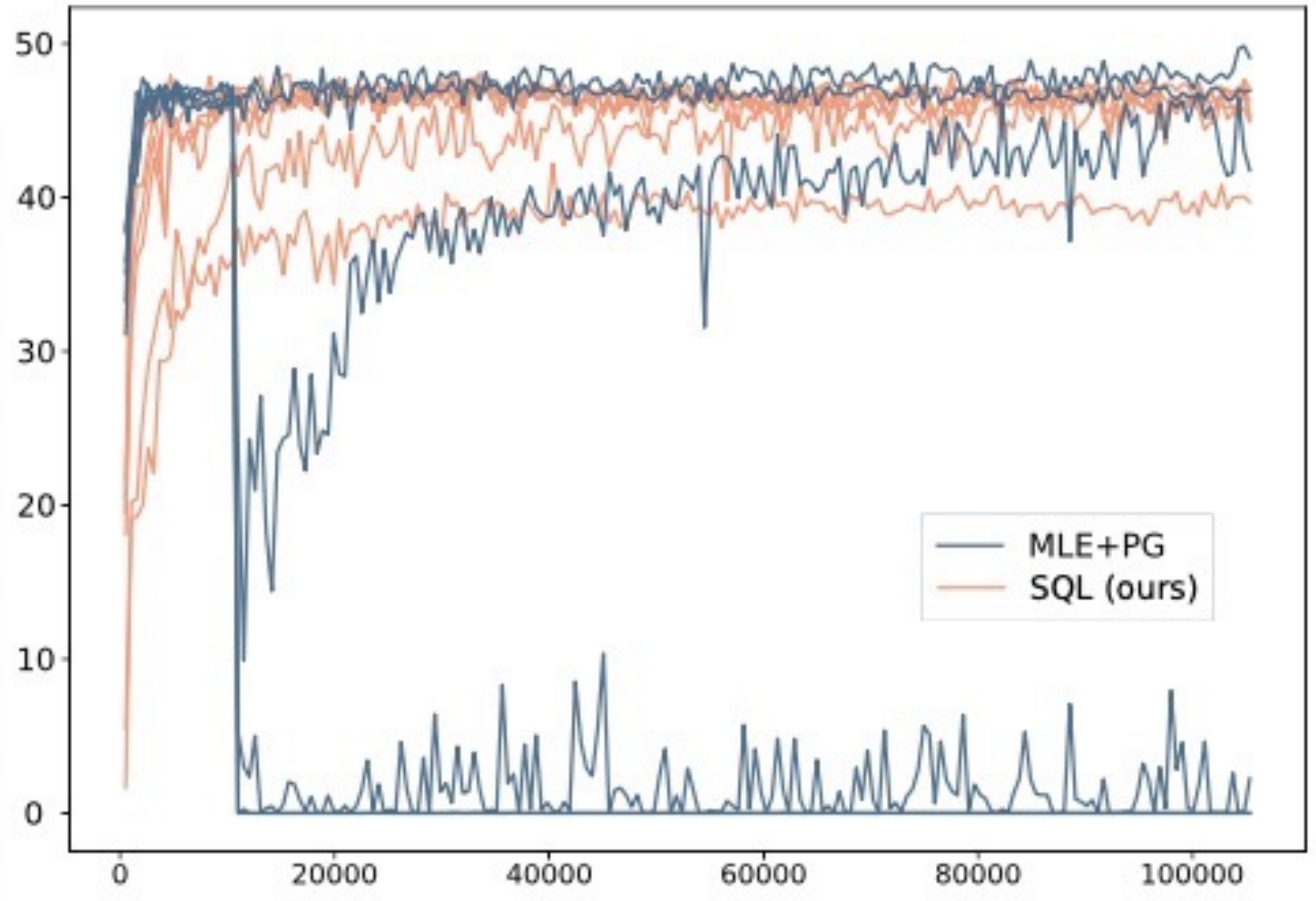
BLEU scores

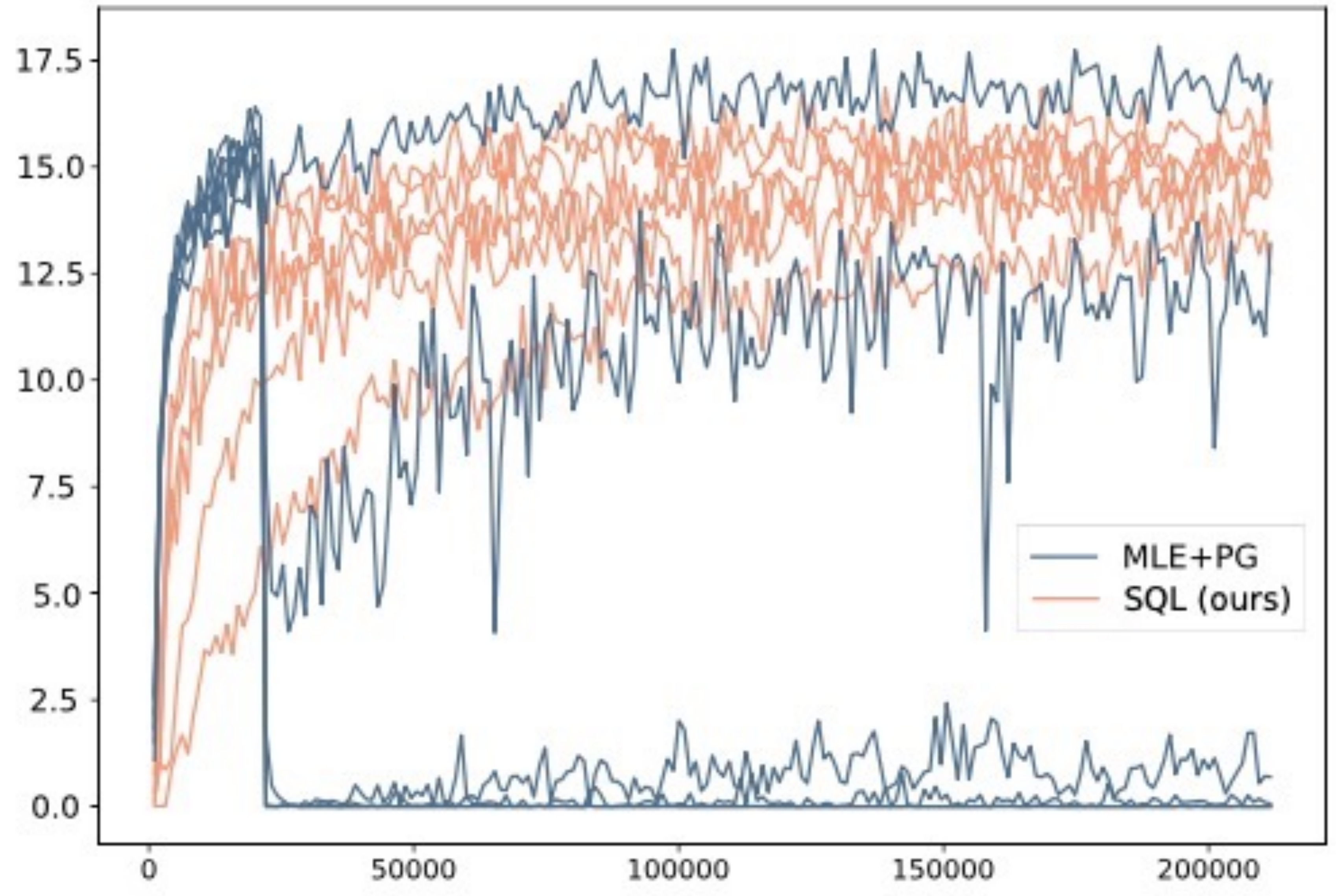


Training curves

Promising results on standard supervised tasks

- **SQL** from scratch is competitive with **MLE** in terms of performance and stability
  - Results on E2E dataset
  - **PG** from scratch fails
- **SQL** is less sensitive to hyperparameters than **MLE+PG**





Training curves of different reward scales

Key Takeaways

- On-policy RL, *e.g., Policy Gradient (PG)*

😈 Extremely low data efficiency

- Off-policy RL, *e.g.,* Q*-learning*

😈 Unstable training; slow updates; sensitive to training data quality

- SQL
  - Objectives based on path consistency

  😇 Combines the best of on-/off-policy, while solving the difficulties

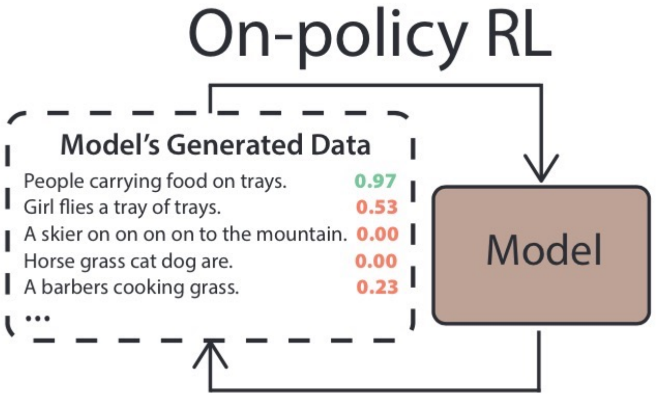
  😇 Stable training from scratch given sparse reward

  😇 Fast updates given large action space

  - Opens up enormous opportunities for integrating more advanced RL for text generation

Questions?

RL for Text Generation: REINFORCE



Given a dataset of input output pairs, $\mathcal{D} \equiv \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)*})\}_{i=1}^{N}$

learn a conditional distribution $p_\theta(\mathbf{y} \mid \mathbf{x})$ that minimizes

expected loss:

$$\mathcal{L}_{\mathrm{RL}}(\boldsymbol{\theta}) = \sum_{(\mathbf{x},\mathbf{y}^*)\in\mathcal{D}} -\sum_{\mathbf{y}\in\mathcal{Y}} p_\theta(\mathbf{y} \mid x)\, r(\mathbf{y}, \mathbf{y}^*)$$

On-policy RL: generate text samples from the current policy p_θ itself

- On-policy exploration to maximize the reward directly

Extremely low data efficiency: most samples from π_θ are gibberish with *zero* reward