DSC190: Machine Learning with Few Labels

Case study: text generation (II)

Zhiting Hu

Lecture 18, November 30, 2021

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Recap: Two Central Goals

- Generating human-like, grammatical, and readable text
  - Model: Progressive generation
  - Learning: Exposure bias, criteria mismatch: reinforcement learning

- 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

Recap: Unsupervised Controlled Generation of Text

- Sentence-level control
  - Text attribute transfer (style transfer) [Hu et al., 2017; Yang et al., 2018]
  - Text content manipulation [Lin et al., 2020]

- Conversation-level control
  - Target-guided Open-domain Conversation [Tang et al., 2019]

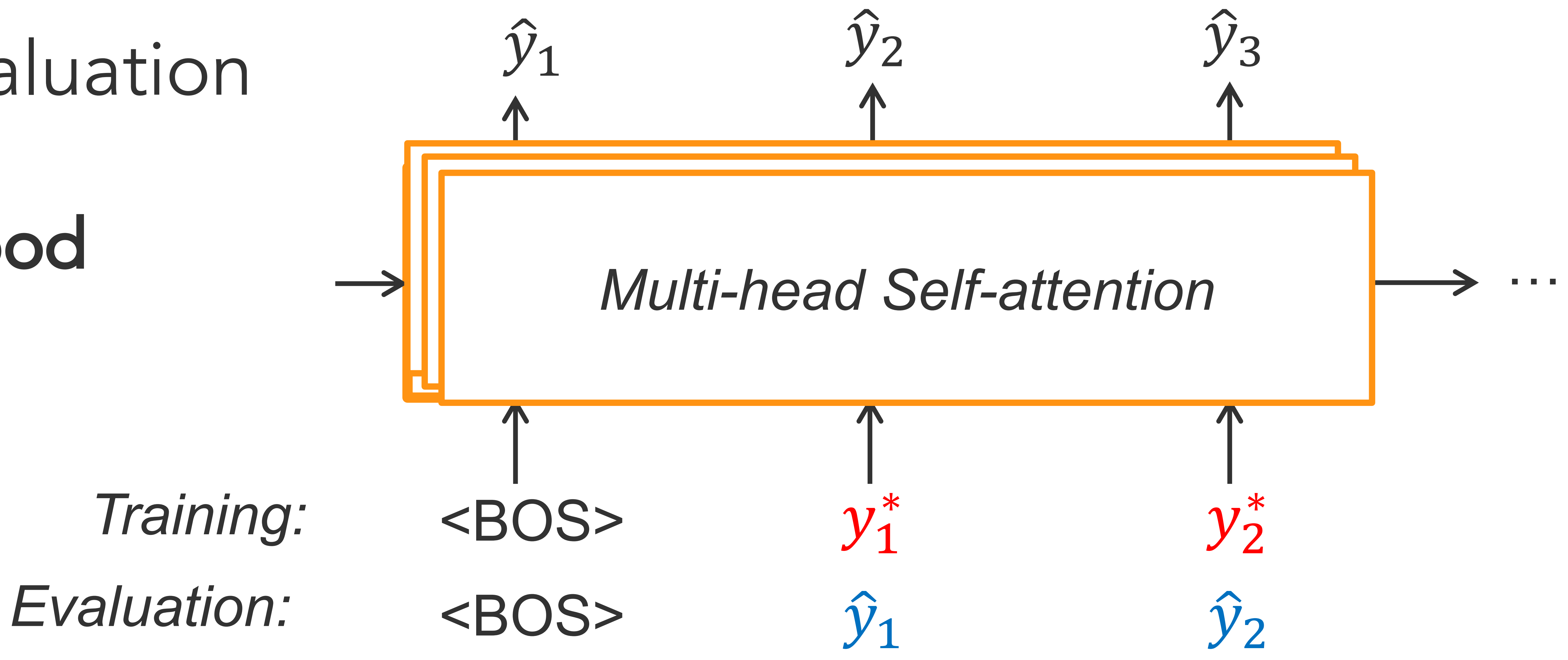
Key idea:
- Decompose the task into competitive sub-objectives
- Use direct supervision for each of the sub-objectives

Recap: Two Issues of MLE

Solution: Reinforcement learning for text generation

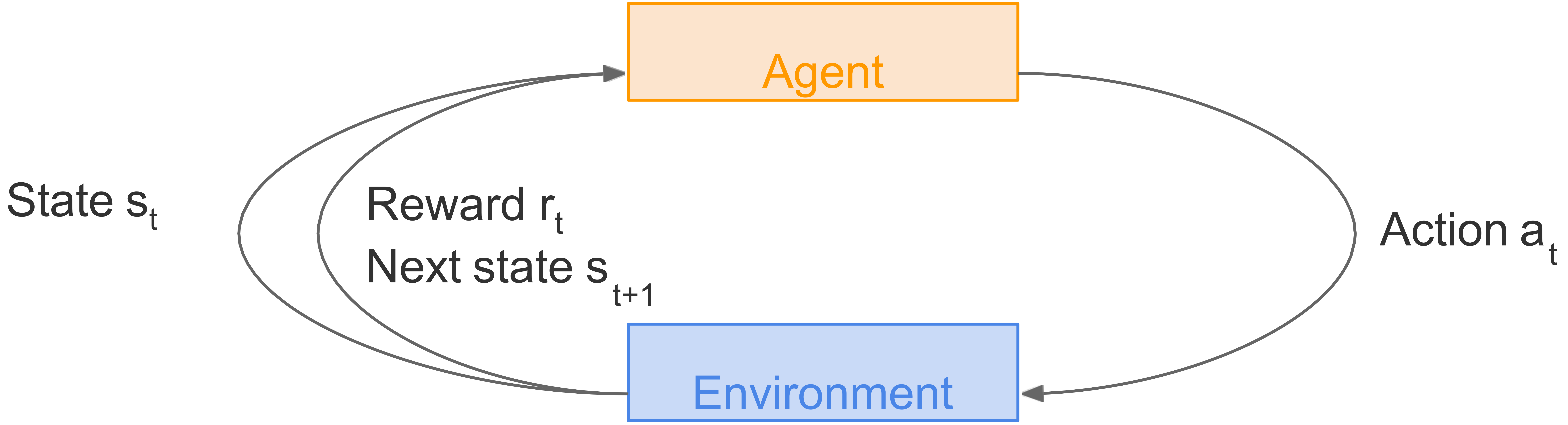
- Exposure bias [Ranzato et al., 2015]
  - **Training**: predict next token given the previous ground-truth sequence
  - **Evaluation**: predict next token given the previous sequence that are generated by the model itself

- 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

Recap: Reinforcement Learning



Recap: Markov Decision Process

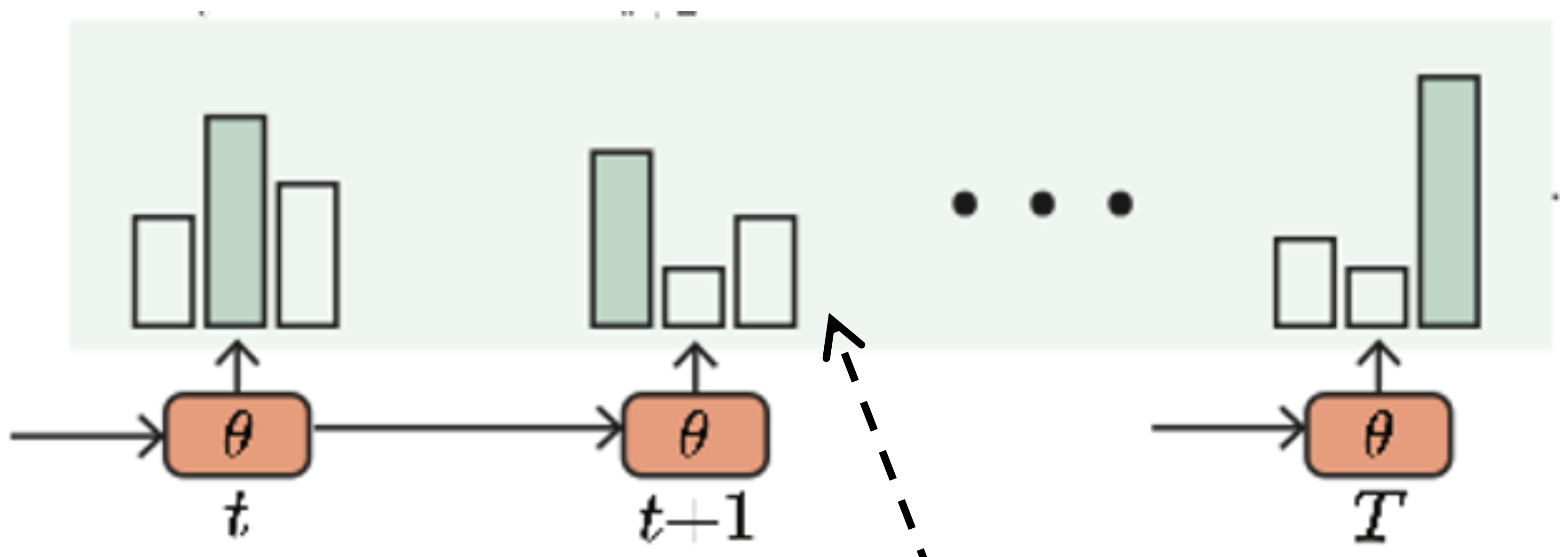
Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor

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: Background



- (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:

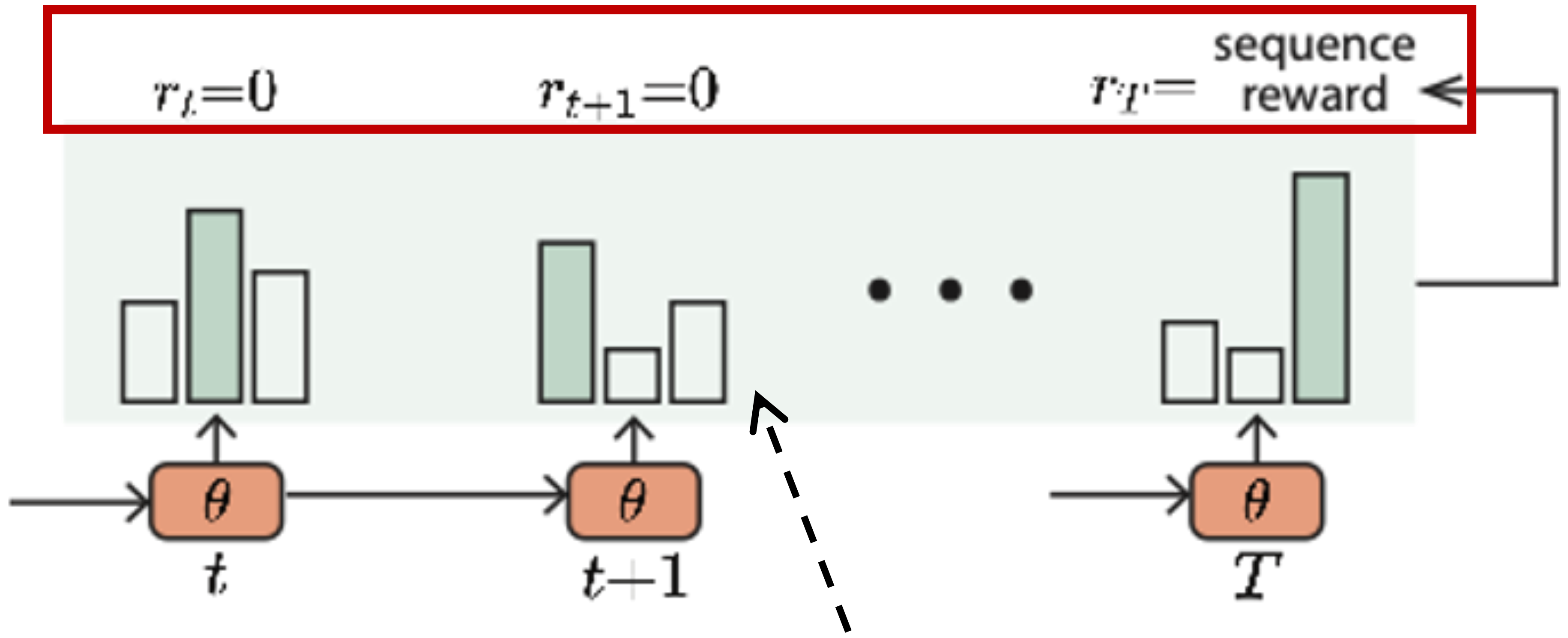
trajectory, τ

action, a_t

state, $\boldsymbol{s}_t$

policy $\pi_\theta(a_t \mid \boldsymbol{s}_t)$

RL for Text Generation: Background



- (Autoregressive) text generation model:

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

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

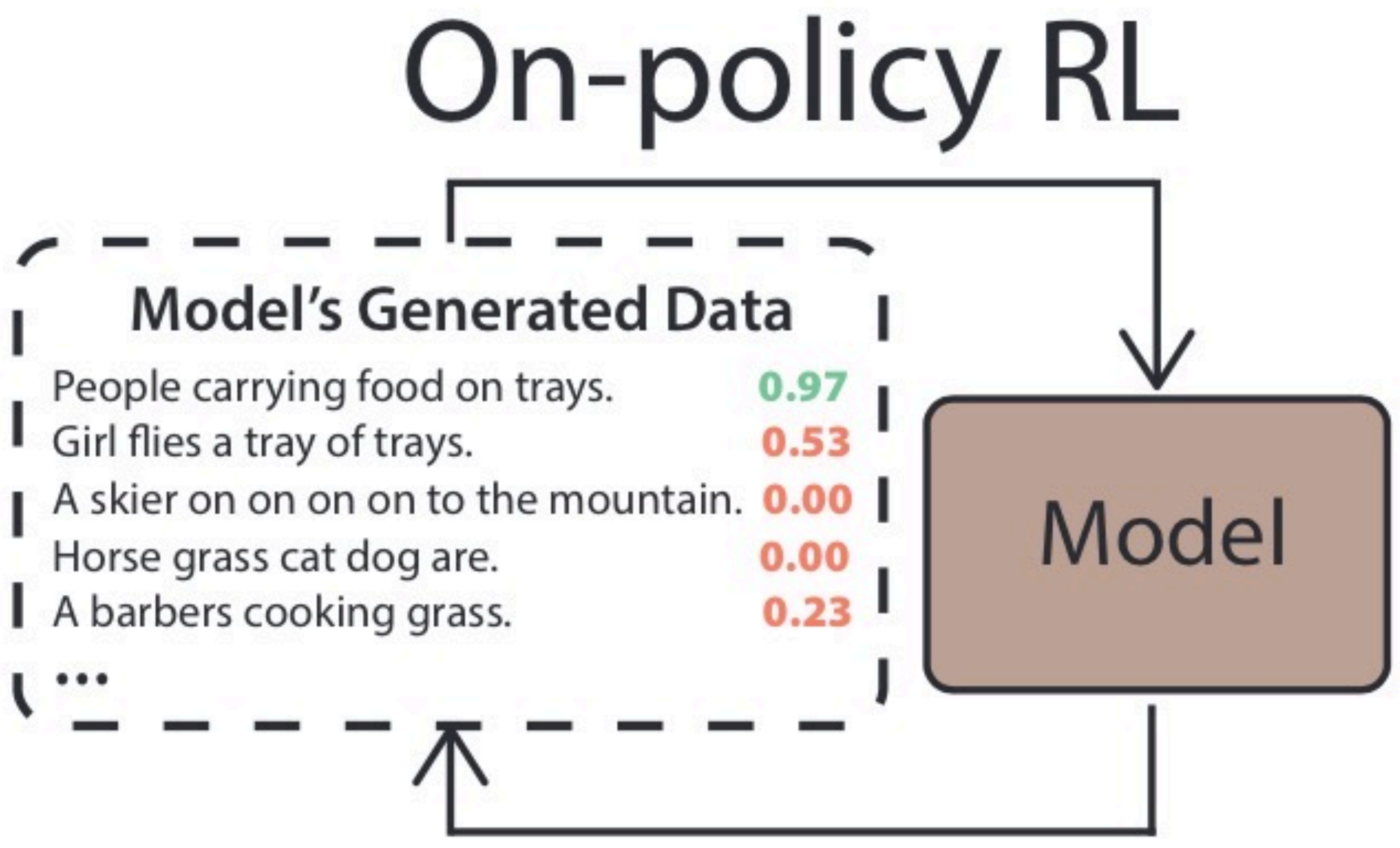
logits

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

- Reward $r_t = r(\boldsymbol{s}_t, a_t)$
  - Often **sparse**: $r_t = 0$ for $t < T$
- The general RL objective: maximize cumulative reward $J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t r_t\right]$
- Q-function: expected *future* reward of taking action a_t in state $\boldsymbol{s}_t$

$$Q^\pi(\boldsymbol{s}_t, a_t) = \mathbb{E}_\pi\left[\sum_{t'=t}^{T} \gamma^{t'} r_{t'} \mid \boldsymbol{s}_t, a_t\right]$$

RL for Text Generation: Background

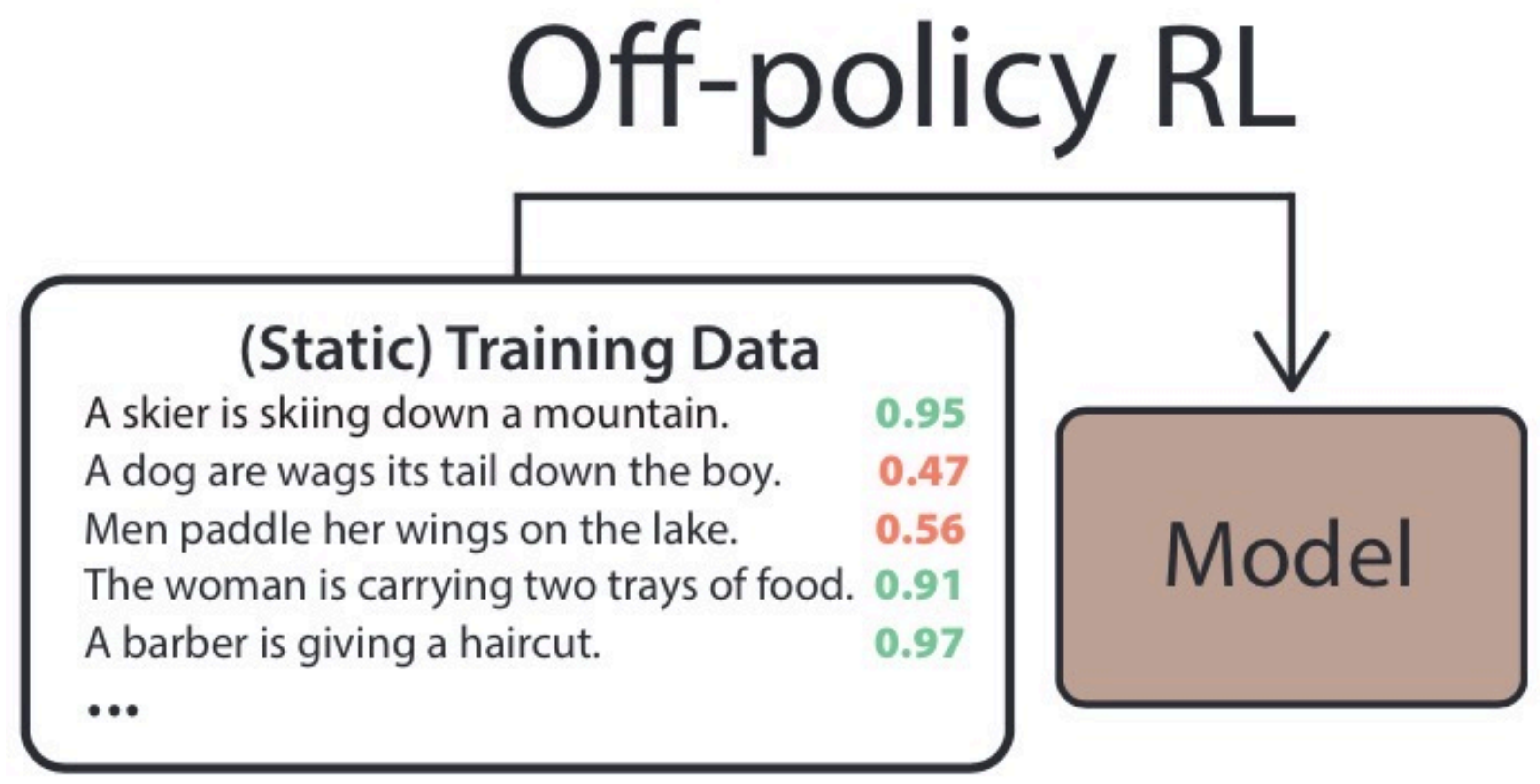


- On-policy RL
  - Most popular, *e.g., Policy Gradient (PG)*

$$\nabla_\theta J(\pi_\theta) = -\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \hat{Q}(\boldsymbol{s}_t, a_t)\nabla_\theta \log \pi_\theta\left(a_t \mid \boldsymbol{s}_t\right)\right]$$

Generate text samples from the current policy π_θ itself
- On-policy exploration to maximize the reward directly

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



RL for Text Generation: Background

- 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

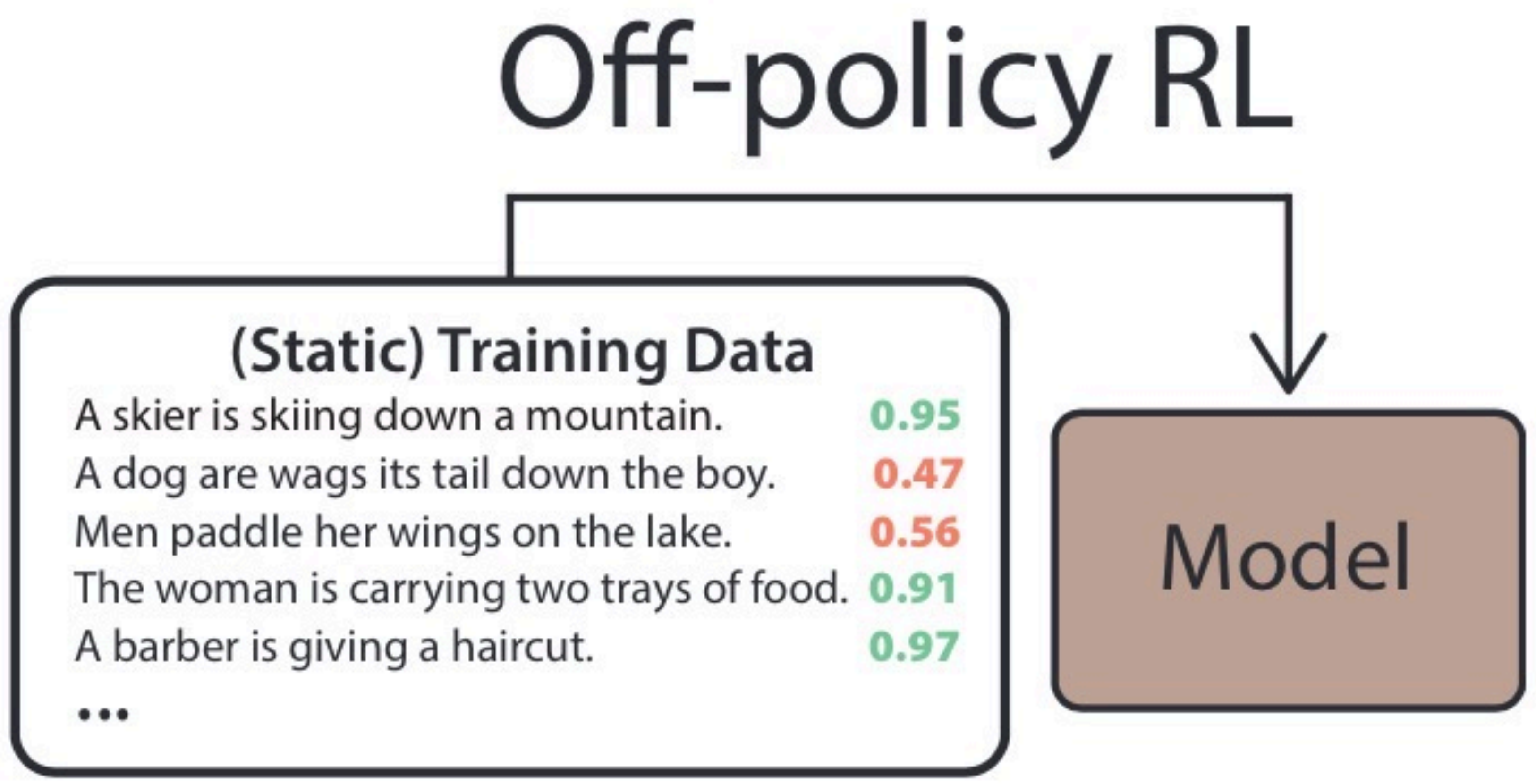
$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\pi'}\left[\frac{1}{2}\left(\boxed{r_t + \gamma \max_{a_{t+1}} Q_{\bar{\theta}}(\boldsymbol{s}_{t+1}, a_{t+1})} - Q_\theta(\boldsymbol{s}_t, a_t)\right)^2\right]$$

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: Background



- 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: Background

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

Extremely low data efficiency

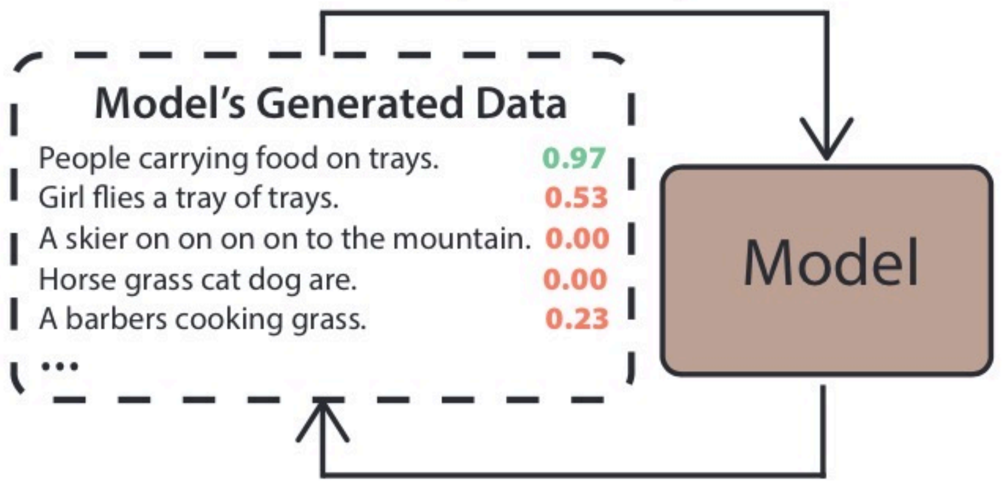
- Off-policy RL, *e.g., Q-learning*

Unstable training due to bootstrapping & sparse reward

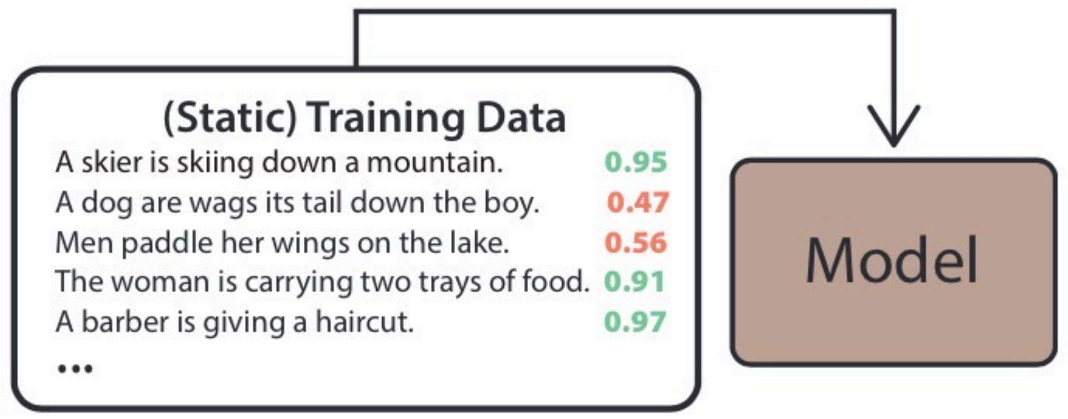
Slow updates due to large action space

Sensitive to training data quality; lacks on-policy exploration

On-policy RL



Off-policy RL



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

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

SQL

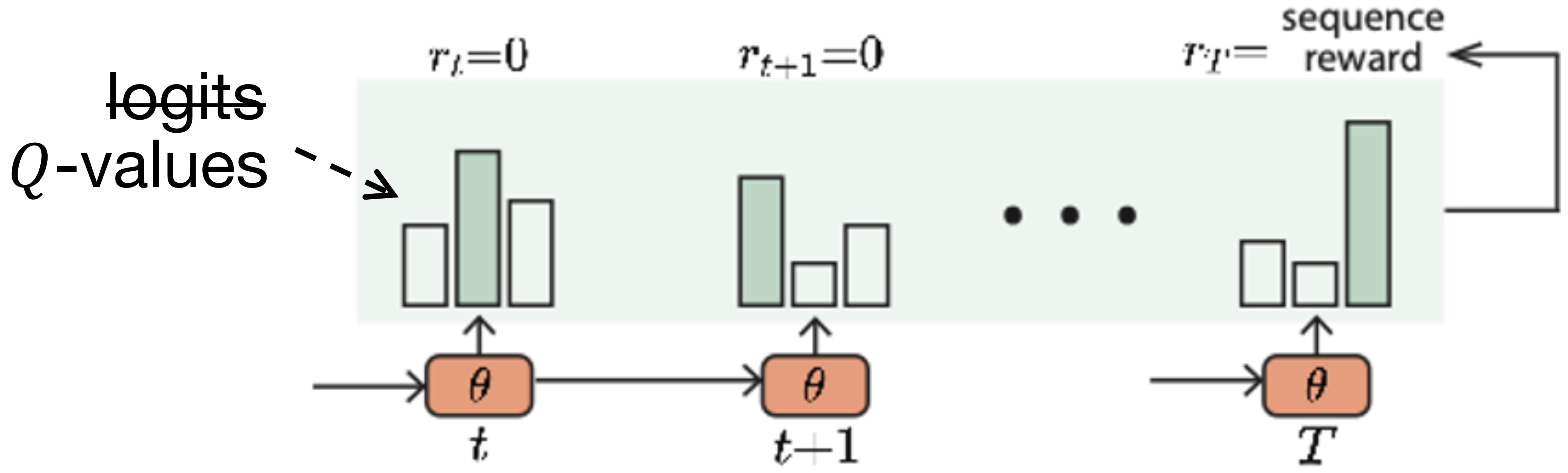
- Goal: entropy regularized

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

- Induced policy

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

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

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

- 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

```python
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
    # ...
```

```python
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")
```

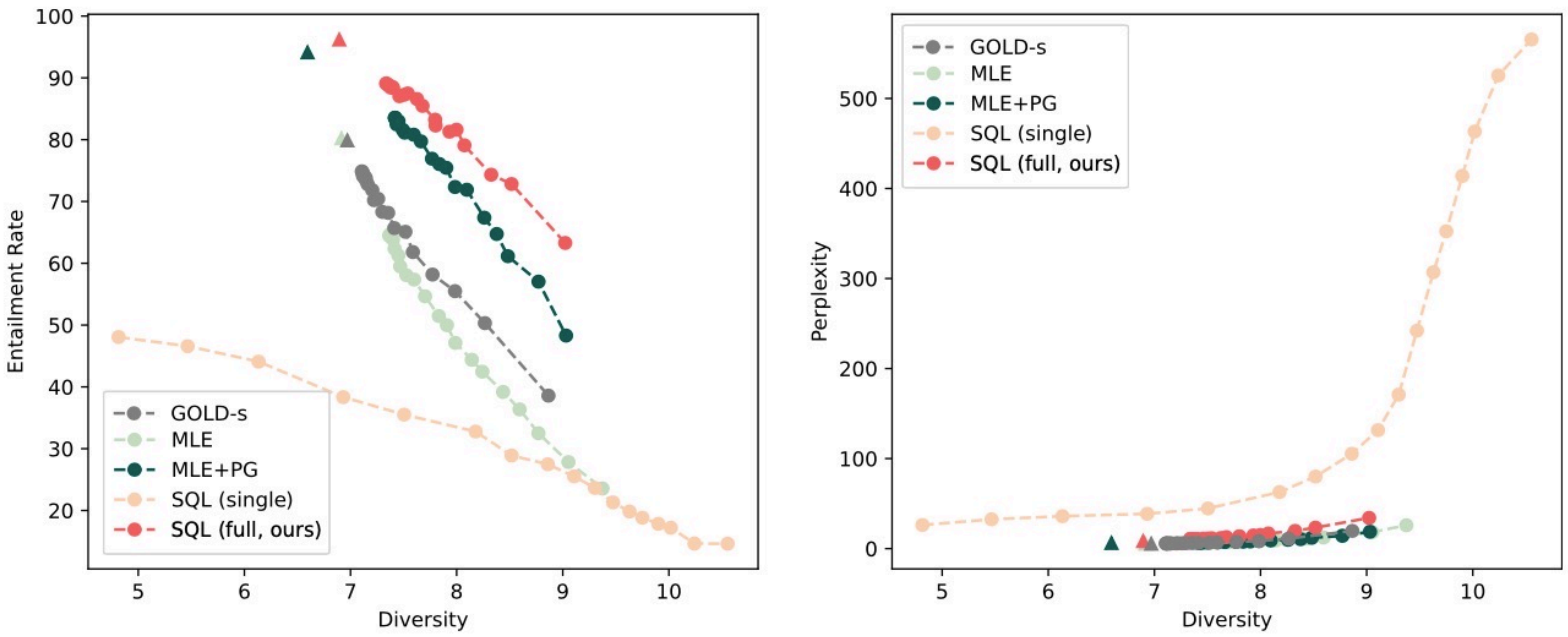
Applications & Experiments

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

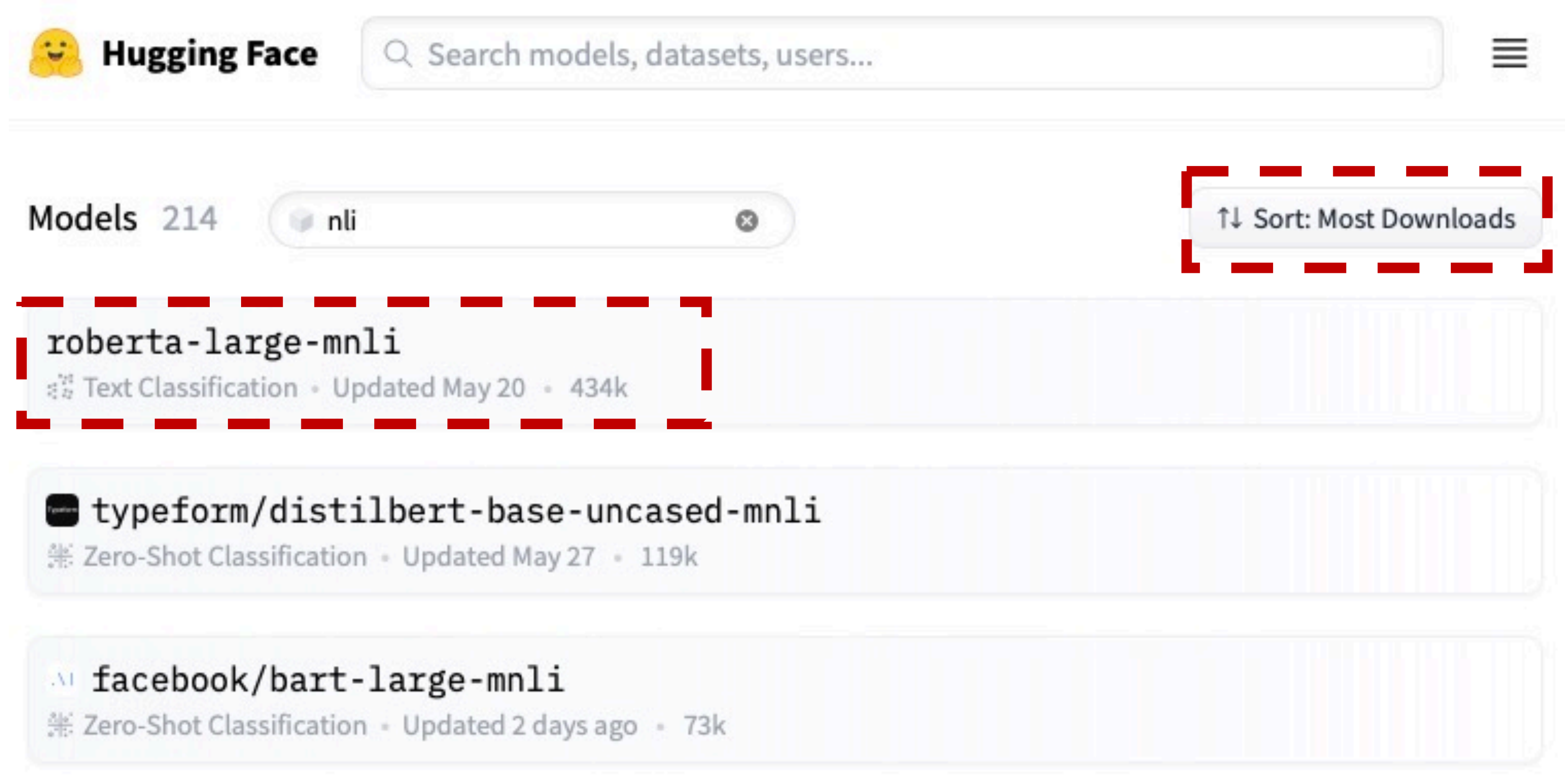
- **MLE** and pure off-policy RL (**GOLD-s**) do not work ← rely heavy on data quality
- **SQL (full)** > **MLE+PG** (PG alone does not work)
- **SQL** (**single**-step only) does not work: the multi-step SQL objective is crucial



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

Application (II): Universal Adversarial Attacks

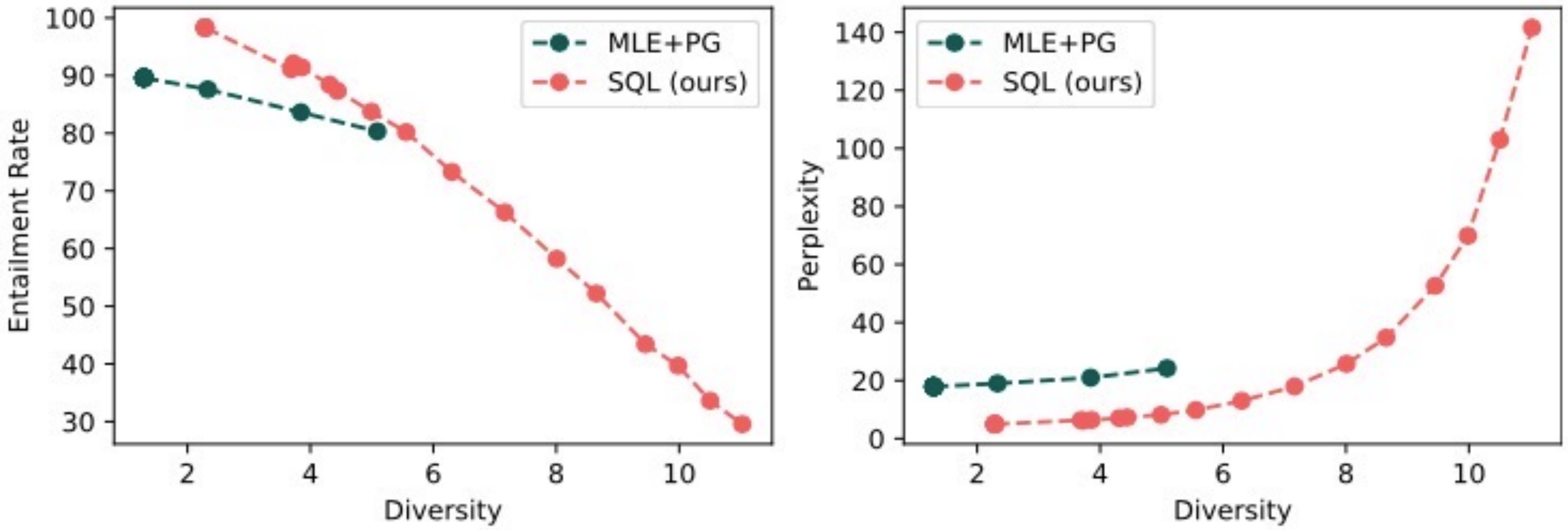
- Attacking entailment classifier
  - Generate **readable** hypotheses that are classified as "entailment" for **all** premises
  - *Unconditional* hypothesis generation model
- Training data:
  - No direct supervision data available
  - "Weak" data: all hypotheses in MultiNLI corpus
- Rewards:
  - Entailment 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

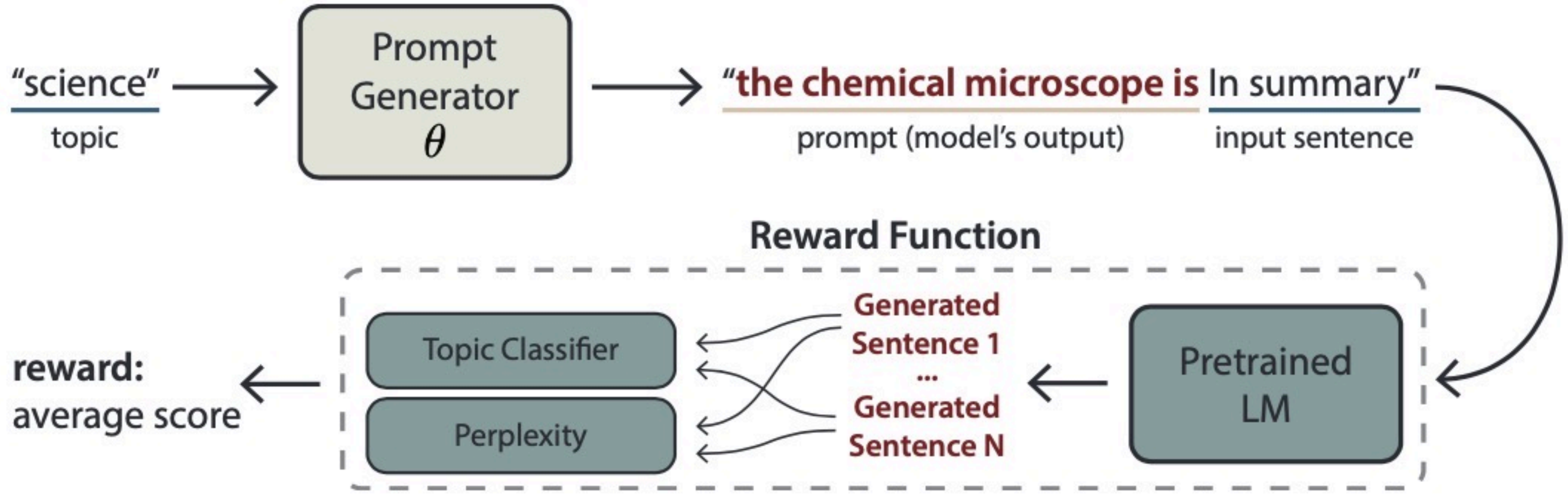


Model	Generation	Rate
MLE+PG	it 's .	90.48
SQL (ours)	the person saint-pierre-et-saint-paul is saint-pierre-et-saint-paul .	97.40

Samples of highest attack rate

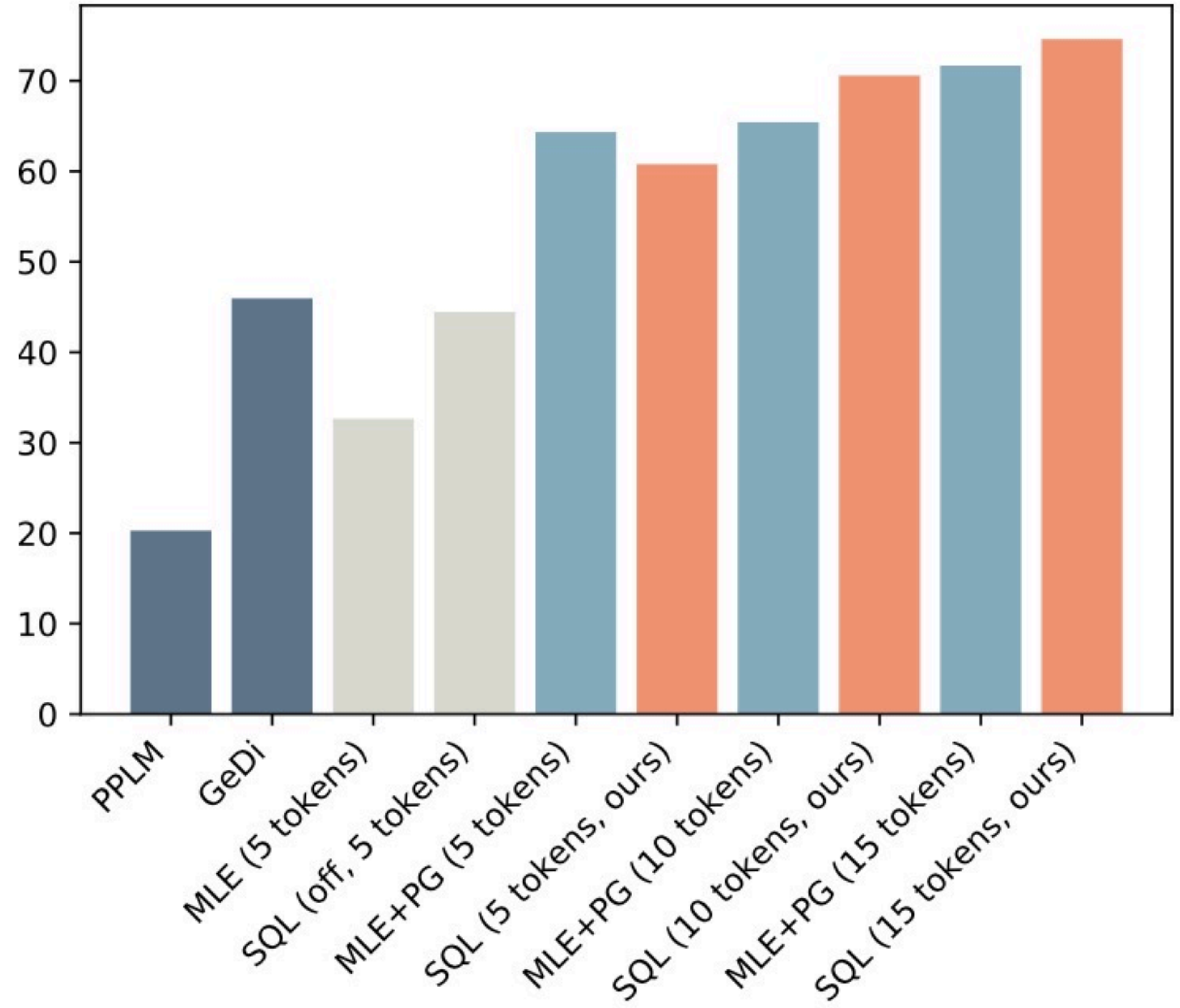
Application (III): Prompt Generation for Controlling LMs

- 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

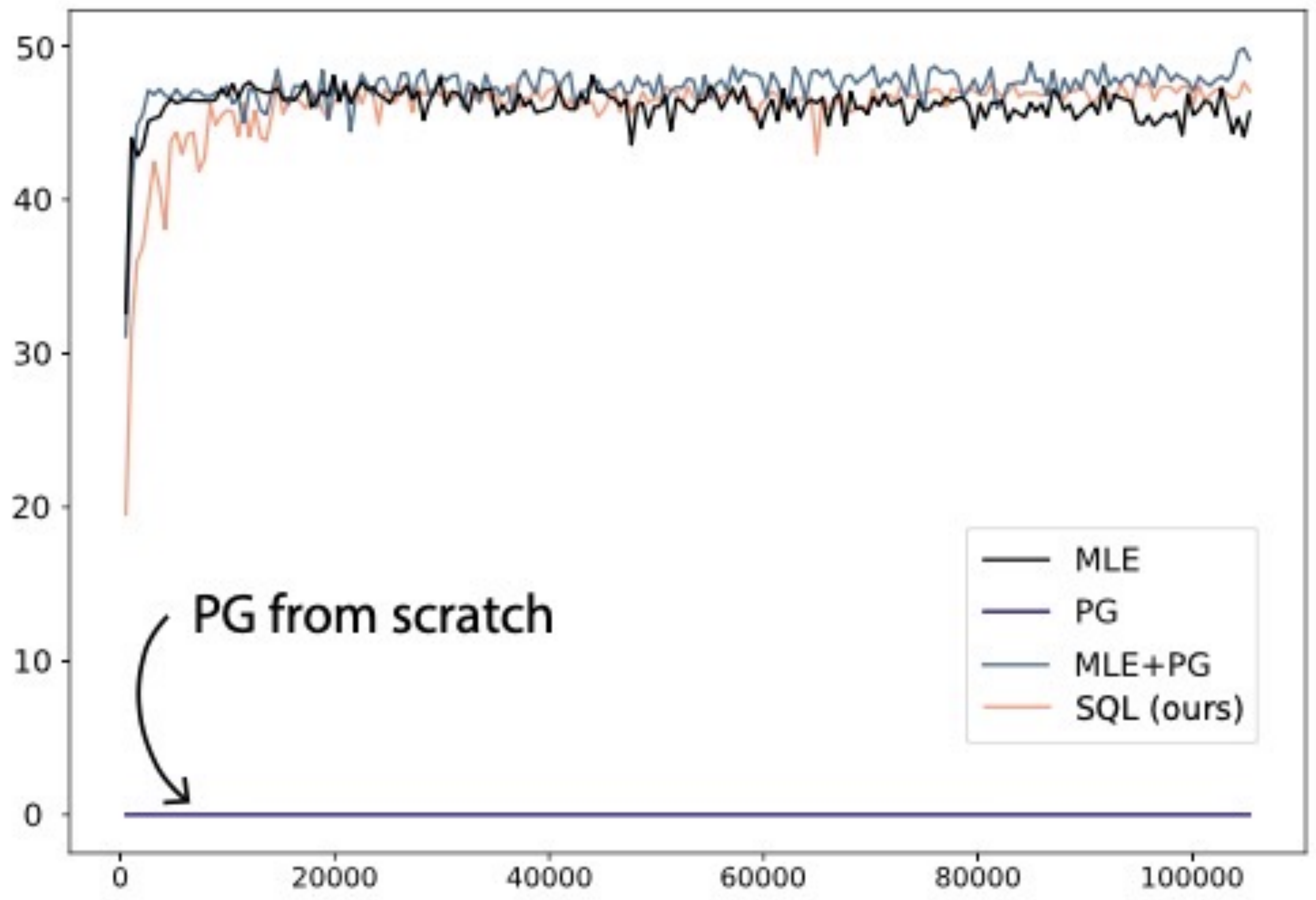
Time cost for generating one sentence

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

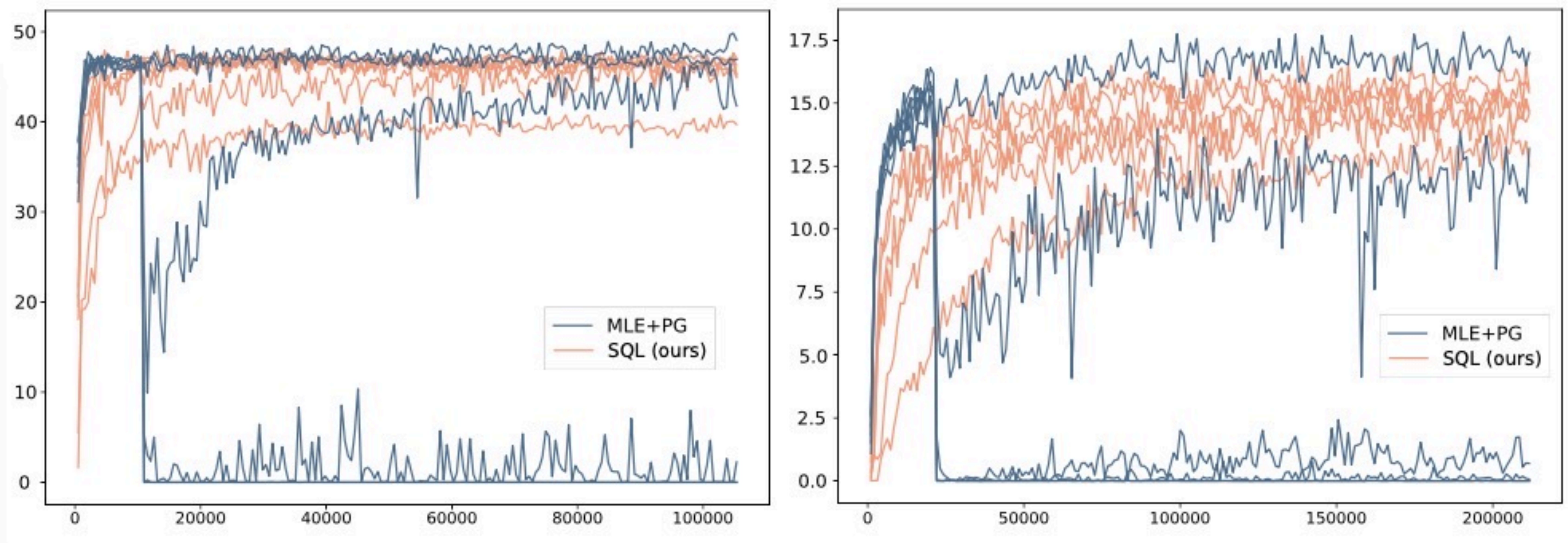
BLEU scores



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

Summary of SQL for Text Generation

- 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?