DSC291: Machine Learning with Few Labels

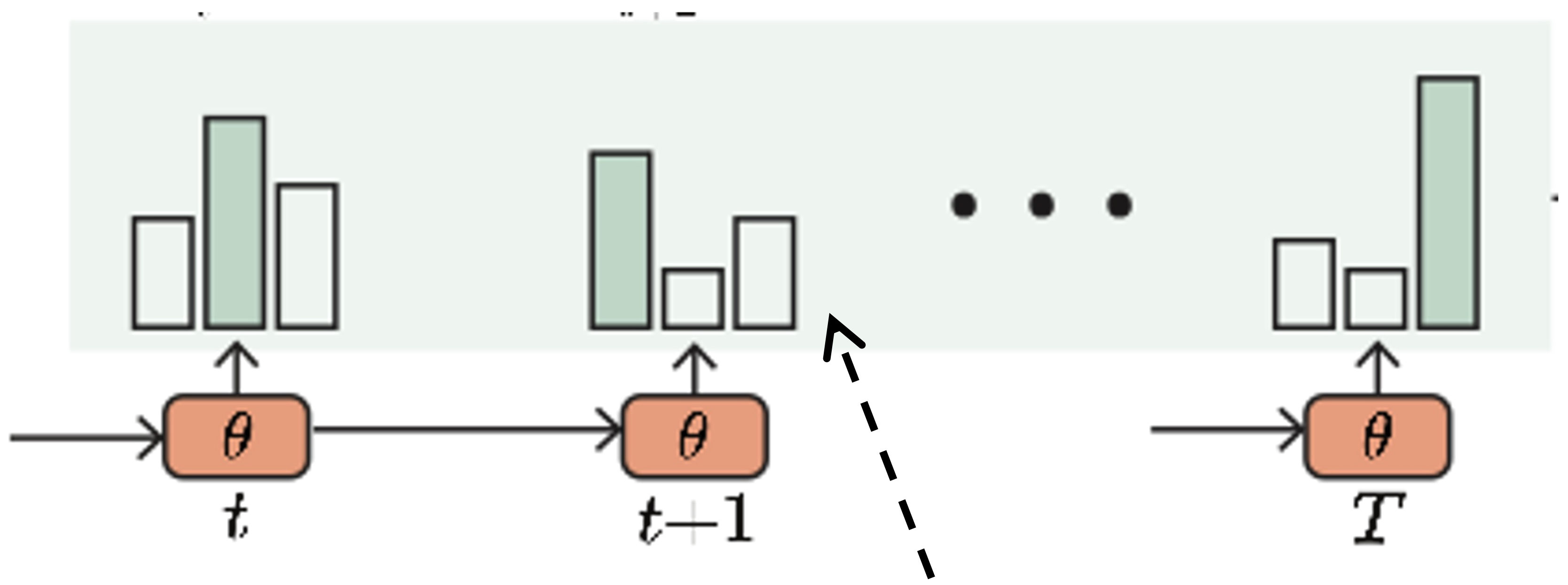
Reinforcement learning for text generation

Zhiting Hu

Lecture 20, February 27, 2023

UC San Diego
HALICIOĞLU DATA SCIENCE INSTITUTE

Recap: RL for Text Generation



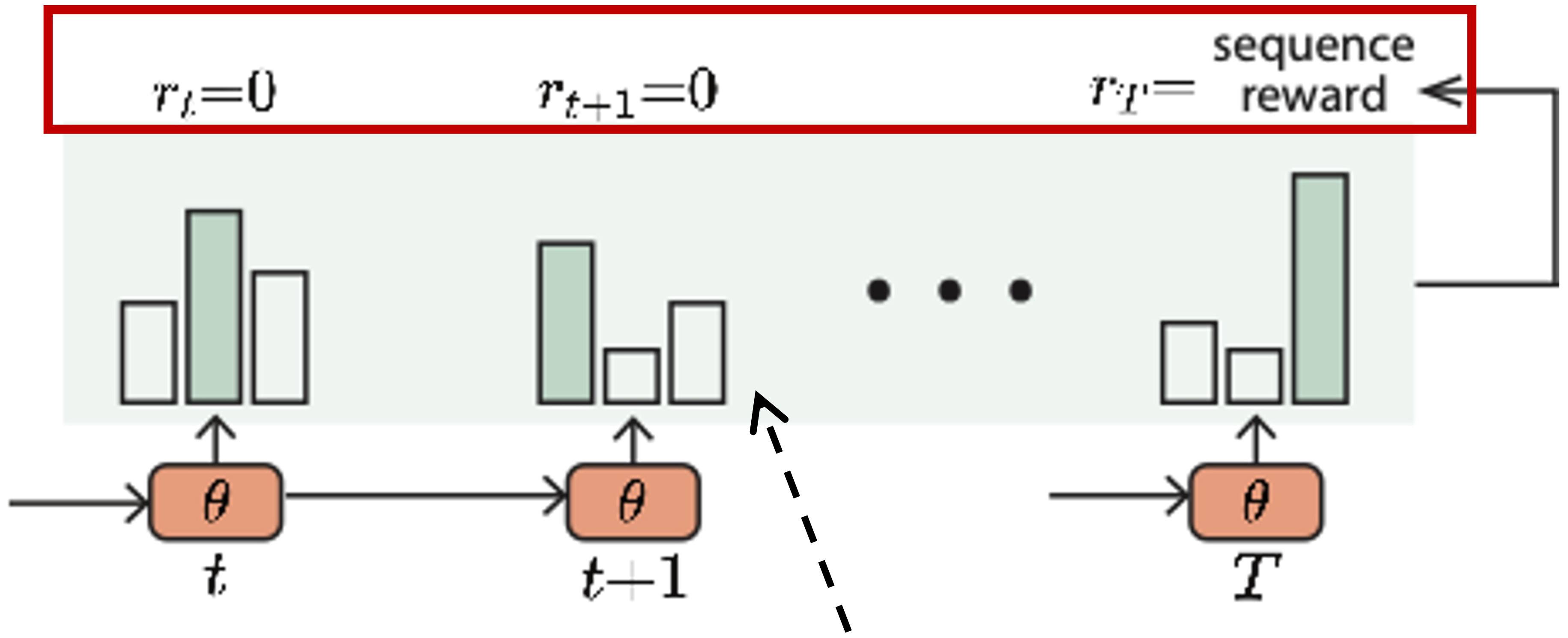
- (Autoregressive) text generation model:

Sentence $\boldsymbol{y} = (y_0, \dots, y_T)$ $\qquad$ $\pi_\theta(y_t \mid \boldsymbol{y}_{<t}) = \mathrm{softmax}(\ f_\theta(y_t|\boldsymbol{y}_{<t})\ )$

logits

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

Recap: RL for Text Generation



- (Autoregressive) text generation model:

Sentence $\boldsymbol{y} = (y_0, \ldots, y_T)$ $\qquad \pi_\theta(y_t \mid \boldsymbol{y}_{<t}) = \text{softmax}(\ f_\theta(y_t | \boldsymbol{y}_{<t})\ )$

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 $\boldsymbol{s}_t$

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

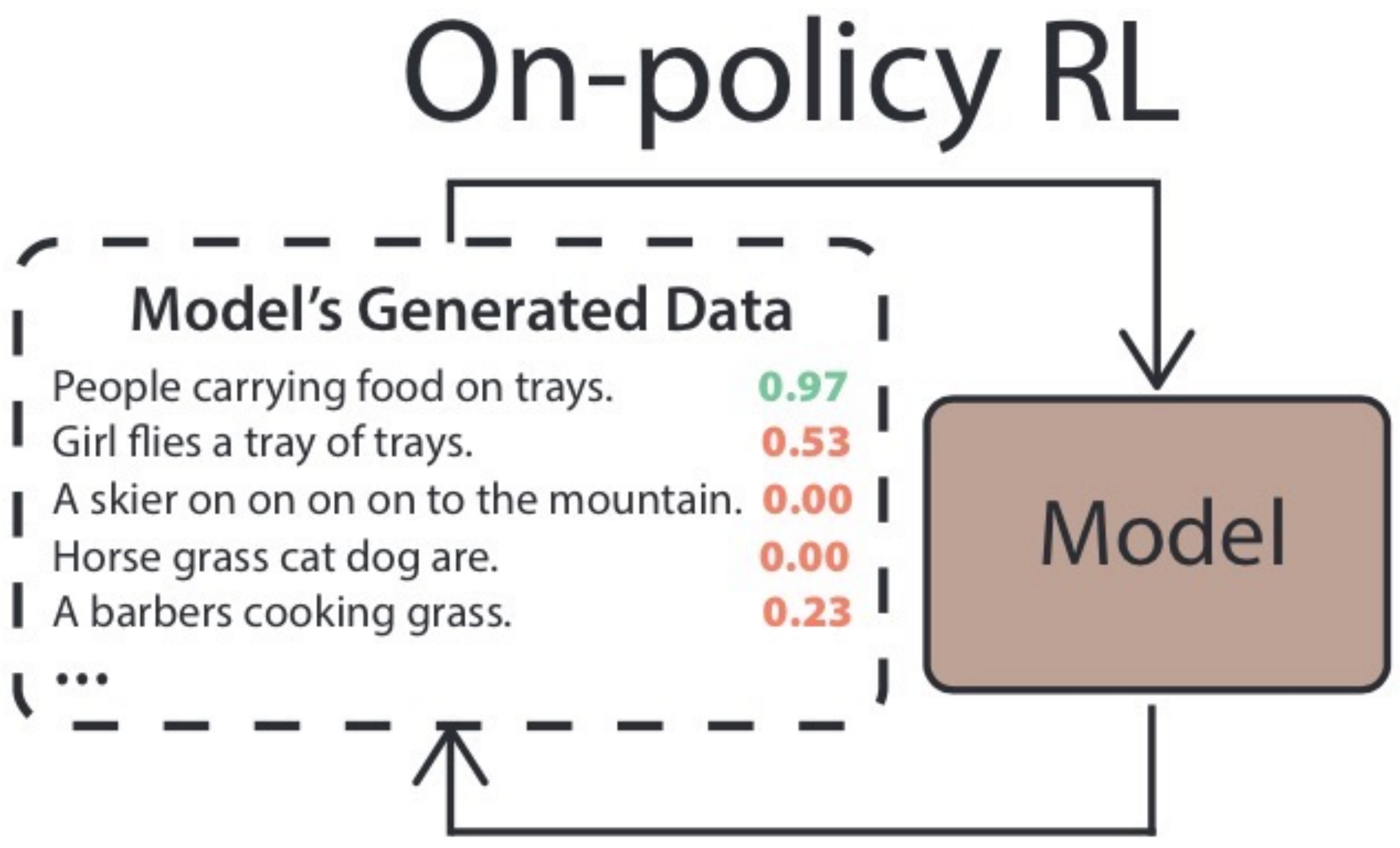
RL for Text Generation: Formulation

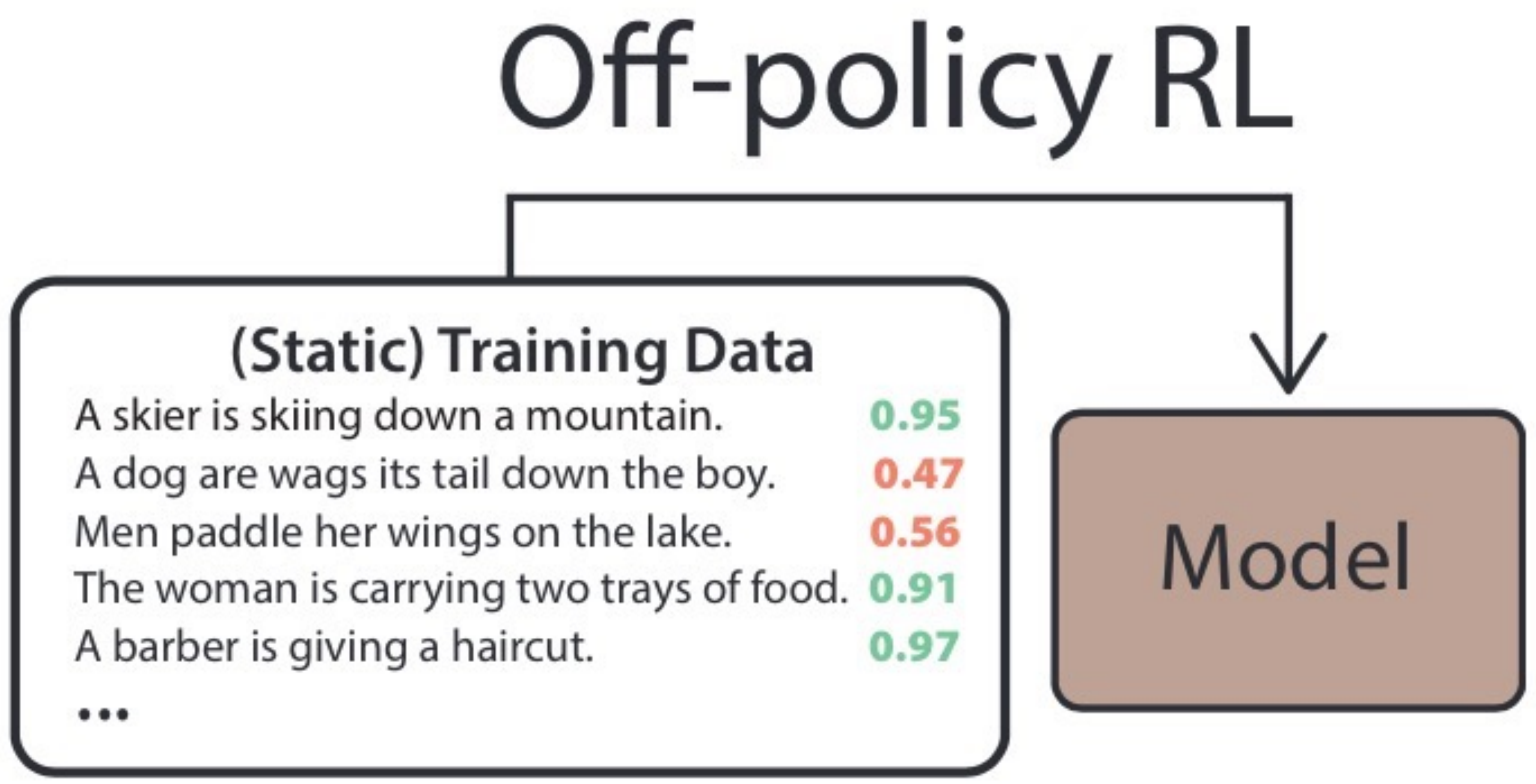
- On-policy RL
  - Most popular, *e.g., Policy Gradient (PG)*

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

Generate text samples from the current policy π_θ itself

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





RL for Text Generation: Formulation

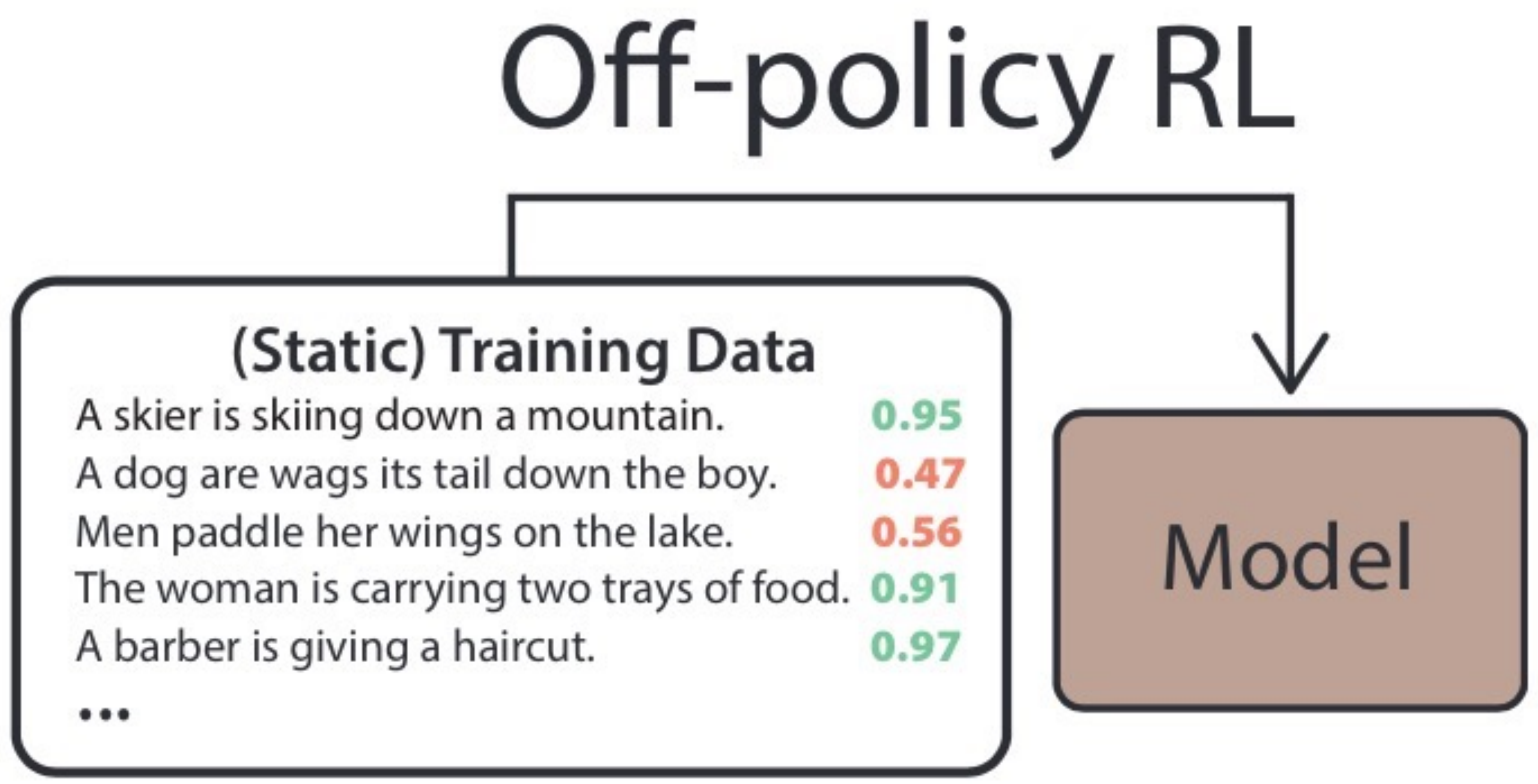
- Off-policy RL
  - *e.g., Q-learning*
  - Implicitly learns the policy π by approximating the $Q^{\pi}(\boldsymbol{s}_t, a_t)$
  - Bellman temporal consistency: $Q^*(\boldsymbol{s}_t, a_t) = r_t + \gamma \max_{a_{t+1}} Q^*(\boldsymbol{s}_{t+1}, a_{t+1})$
  - Learns Q_θ with the regression objective:

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

Arbitrary policy

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

RL for Text Generation: Formulation



- 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:

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

Arbitrary policy

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

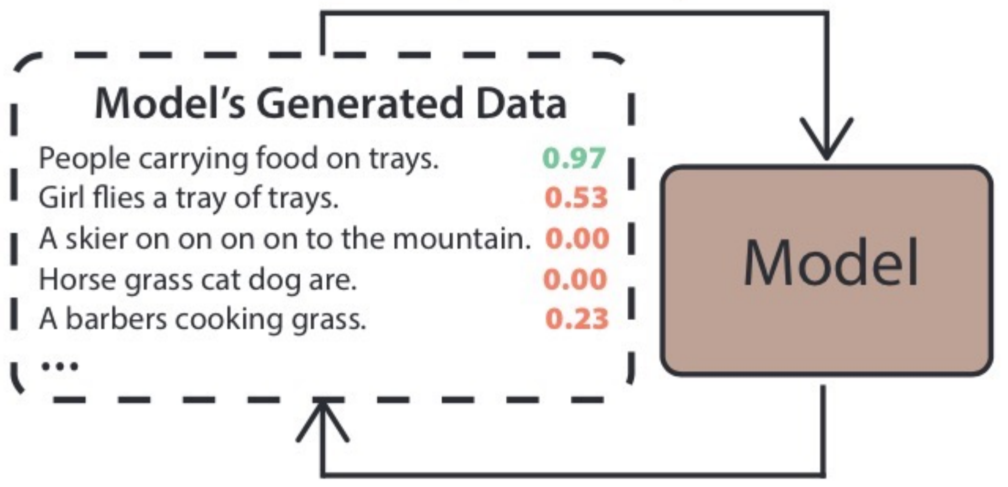
- After learning, induces the policy as $a_t = \text{argmax}_a\, Q_{\theta^*}(\boldsymbol{s}_t, a)$

RL for Text Generation: Formulation

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

Extremely low data efficiency

On-policy RL



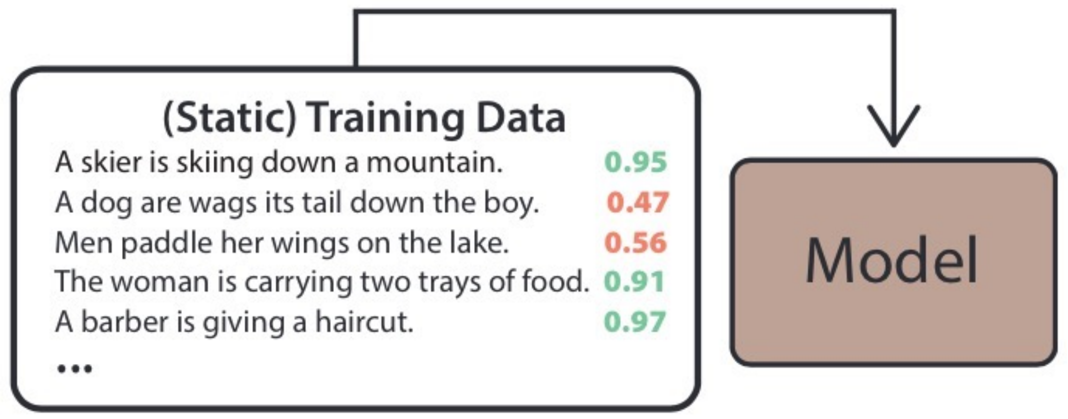
- Off-policy RL, *e.g., Q-learning*

Unstable training due to bootstrapping & sparse reward

Slow updates due to large action space

Sensitive to off-policy data quality

Off-policy RL



… Limited success for training text generation

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

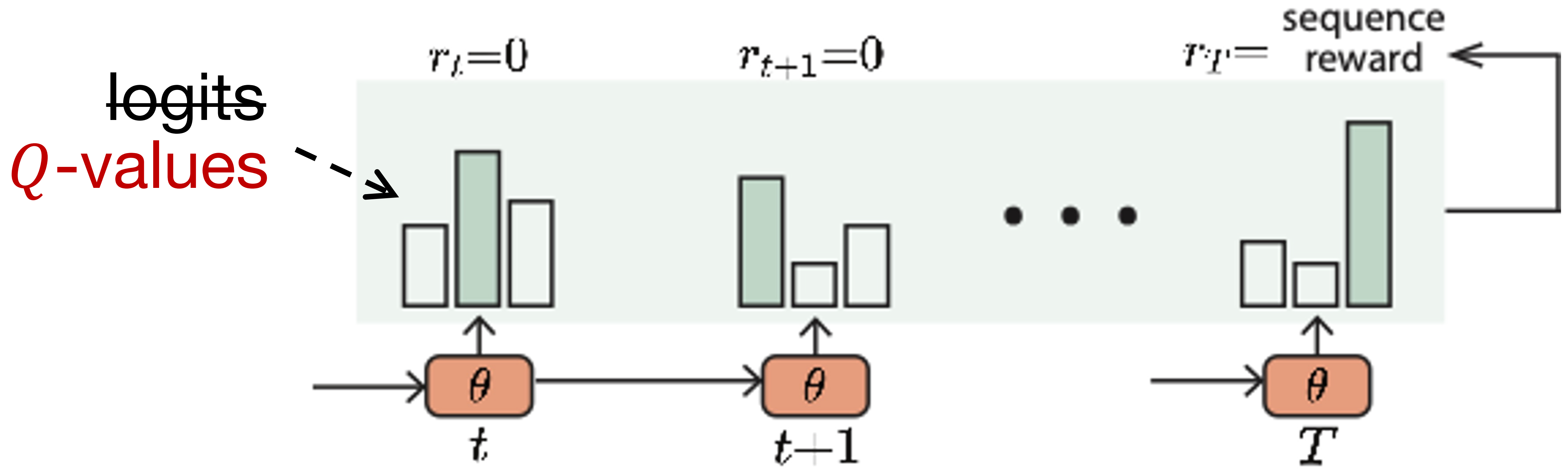
- Goal: entropy regularized

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

- Induced policy

$$\pi_{\theta^*}(a_t \mid \boldsymbol{s}_t) = \mathrm{softmax}\left(Q_{\theta^*}(a_t | \boldsymbol{s}_t) \right)$$

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

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

- 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

$$\pi_{\theta^*}(a_t \mid \boldsymbol{s}_t) = \text{softmax}(\, Q_{\theta^*}(a_t | \boldsymbol{s}_t) \,)$$

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

😇 Stable / efficient

Efficient Training via Path Consistency

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

$$\pi^*(a \mid \boldsymbol{s}) = \operatorname{softmax}(\, Q^*(a \mid \boldsymbol{s})\,)$$

- (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

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

[Nachum et al., 2017]

Implementation is easy

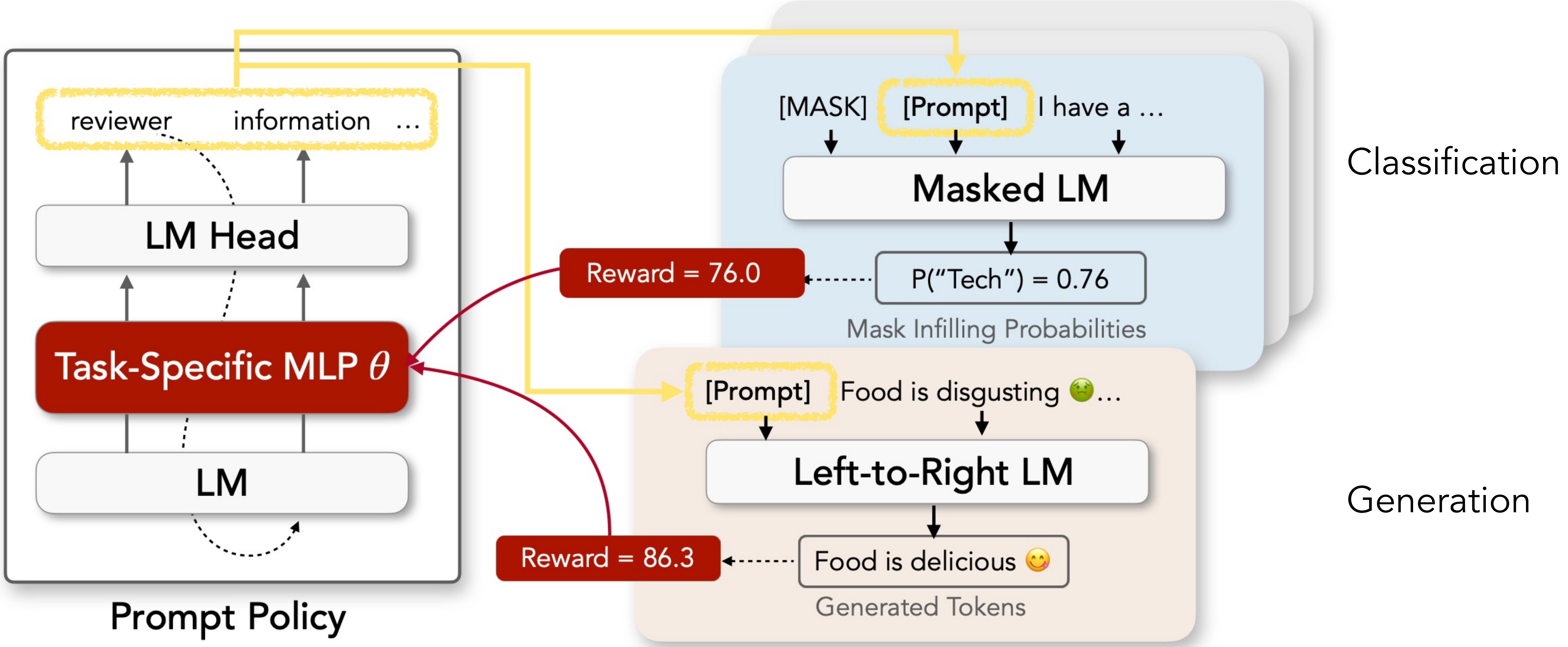
```python
model = TransformerLM(...)

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): Prompt Optimization for Controlling LMs

- Optimize discrete prompts to steer pretrained LMs to produce desired outputs



Application (I): Prompt Optimization for Controlling LMs

- Optimize discrete prompts to steer pretrained LMs to produce desired outputs

Methods	Frozen LMs	Automated	Gradient-free	Guided Optimize	Few-shot	Zero-shot	Transferrable b/w LMs	Interpret.
Finetuning	✗	✓	✗	✓	✗	✗	✗	✗
In-context Demo.	✓	✓	✓	✗	✓	✗	✓	✓
Instructions	✓	✗	✓	✗	✓	✓	✓	✓
Manual Prompt	✓	✗	✓	✗	✓	✓	✓	✓
Soft Prompt Tuning	✓	✓	✗	✓	✓	✗	✗	✗
Discrete Prompt Enum.	✓	✓	✓	✗	✓	✓	✓	✓
AutoPrompt	✓	✓	✗	✓	✓	✗	✓	✓
RLPrompt (**Ours**)	✓	✓	✓	✓	✓	✓	✓	✓

Comparison of different (prompting) paradigms for using pretrained LMs on downstream tasks, in terms of a number of desirable properties.

Application (I): Prompt Optimization for Controlling LMs

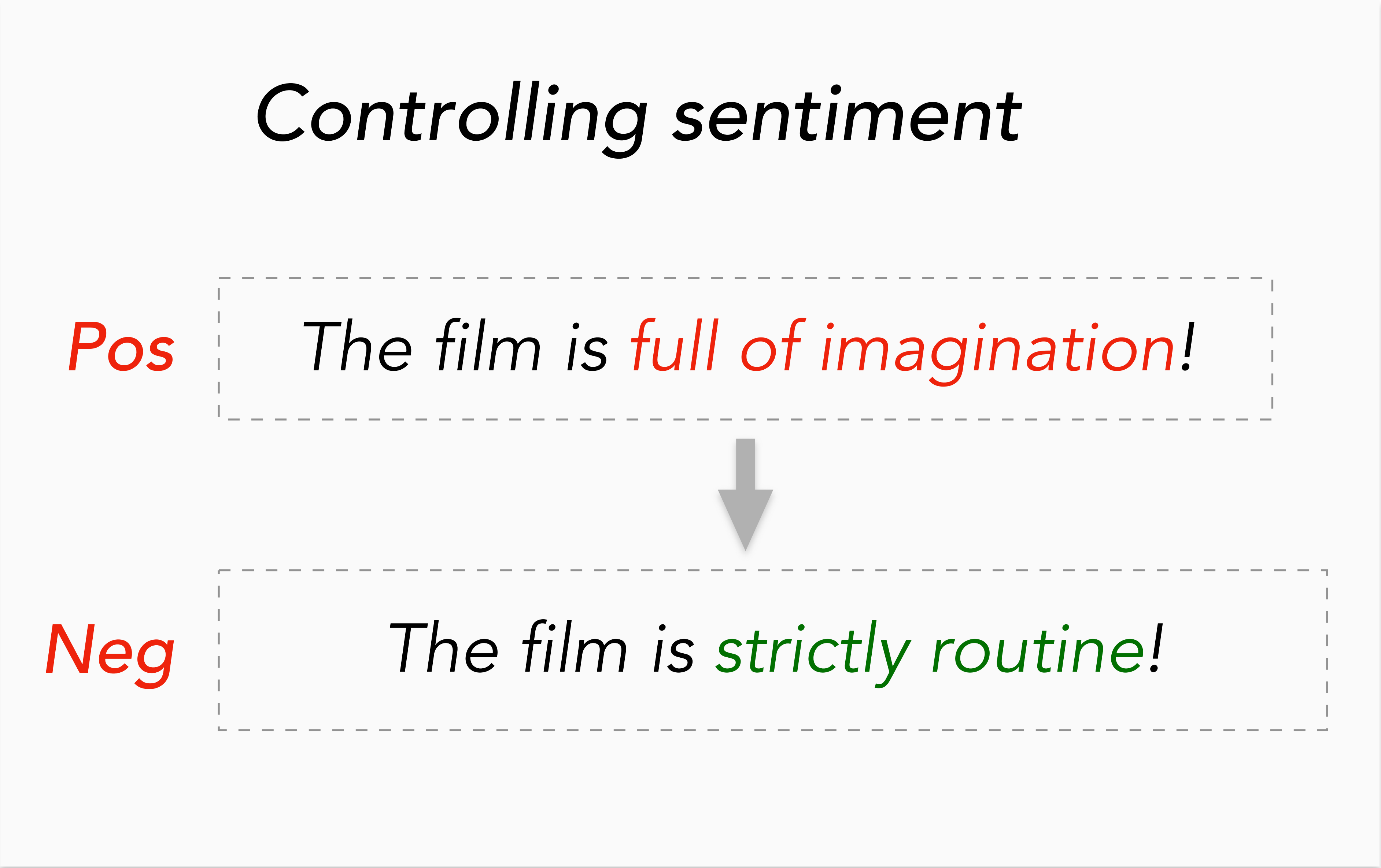
- Few-shot classification

	SST-2	Yelp P.	MR	CR	AG's News
Finetuning	80.6 (3.9)	88.7 (4.7)	67.4 (9.7)	73.3 (7.5)	**84.9** (3.6)
Manual Prompt	82.8	83.0	80.9	79.6	76.9
In-context Demo.	85.9 (0.7)	89.6 (0.4)	80.6 (1.4)	85.5 (1.5)	74.9 (0.8)
Instructions	89.0	84.4	85.2	80.8	54.8
Prompt Tuning *(Soft Prompt Tuning)*	73.8 (10.9)	88.6 (2.1)	74.1 (14.6)	75.9 (11.8)	82.6 (0.9)
Black-Box Tuning *(Mixed Prompt + Soft Tuning)*	89.1 (0.9)	93.2 (0.5)	86.6 (1.3)	**87.4** (1.0)	83.5 (0.9)
GrIPS *(Discrete Prompt Enum.)*	87.1 (1.5)	88.2 (0.1)	86.1 (0.3)	80.0 (2.5)	65.4 (9.8)
AutoPrompt	75.0 (7.6)	79.8 (8.3)	62.0 (0.8)	57.5 (5.8)	65.7 (1.9)
RLPrompt (Ours)	**90.1** (1.8)	**93.9** (1.8)	**86.7** (2.4)	87.2 (1.7)	77.2 (2.0)

Table 3: Results of few-shot text classification, comparing with methods of different paradigms in Table 1

Application (I): Prompt Optimization for Controlling LMs

- Text style transfer



Application (I): Prompt Optimization for Controlling LMs

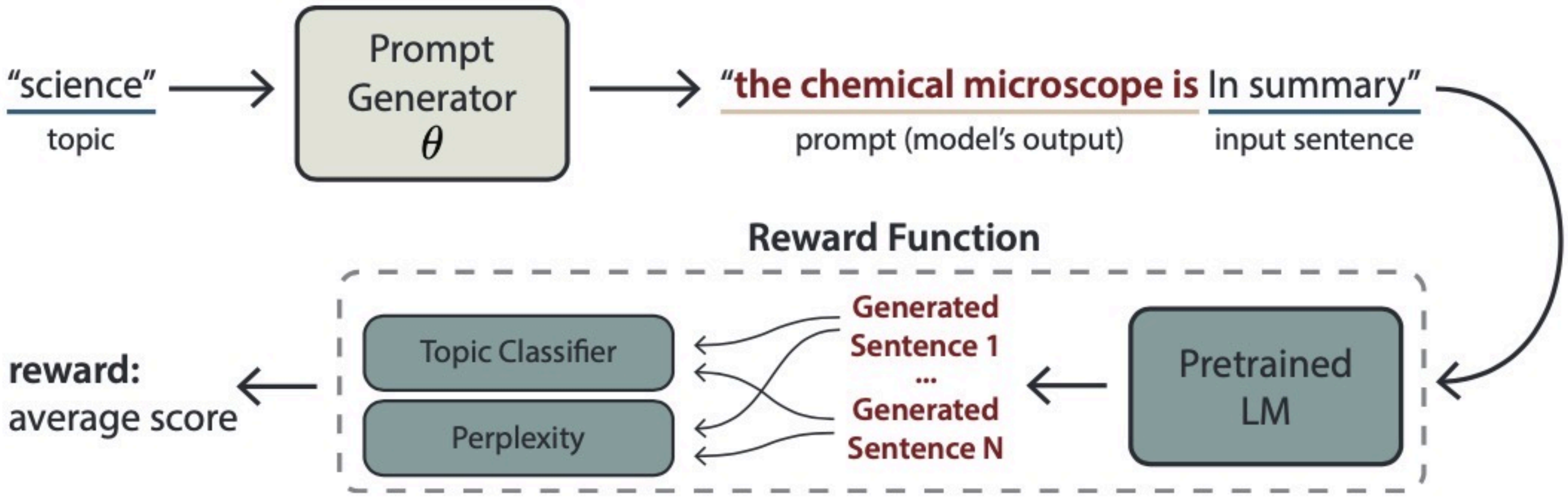
- Text style transfer

Model	Content	Style	Fluency	$J(\text{C, S, F})$	GM(C, S, F)	BLEU	BERTScore	PPL↓
Oracles								
Copy	100 (0.0)	1.4 (0.0)	92.2 (0.0)	11.9 (0.0)	23.5 (0.0)	30.1 (0.0)	62.2 (0.0)	20.6 (0.0)
Reference	62.2 (0.0)	78.9 (0.0)	88.7 (0.0)	55.9 (0.0)	75.8 (0.0)	100 (0.0)	100 (0.0)	30.8 (0.0)
Training Baselines								
Style Transformer	75.2 (0.1)	96.4 (0.1)	58.6 (0.2)	46.1 (0.2)	75.2 (0.1)	27.6 (0.1)	56.1 (0.0)	78.2 (0.3)
DiRR	**78.8 (0.0)**	**97.7 (0.1)**	75.6 (0.2)	59.6 (0.2)	83.5 (0.1)	**30.0 (0.0)**	**61.7 (0.0)**	40.6 (0.1)
Prompting Baselines (GPT-2 xlarge)								
Null Prompt	37.4 (0.1)	94.8 (0.1)	97.6 (0.1)	33.6 (0.1)	70.2 (0.1)	6.6 (0.1)	35.8 (0.1)	59.5 (2.0)
Random Prompt	39.6 (0.1)	93.8 (0.2)	**97.8 (0.1)**	34.7 (0.2)	71.3 (0.1)	7.3 (0.1)	37.4 (0.1)	60.5 (1.6)
Manual Prompt	64.2 (1.0)	91.5 (0.6)	93.2 (0.2)	53.4 (1.2)	81.8 (0.5)	19.2 (0.6)	53.1 (0.8)	35.5 (1.4)
RLPROMPT ***(Ours)***								
distilGPT-2	57.3 (0.3)	96.5 (0.1)	85.3 (0.3)	46.0 (0.2)	77.9 (0.1)	15.7 (0.1)	49.1 (0.1)	43.6 (0.6)
GPT-2 small	60.0 (0.1)	96.4 (0.1)	89.0 (0.5)	50.7 (0.3)	80.1 (0.1)	16.5 (0.1)	51.3 (0.1)	37.8 (0.9)
GPT-2 medium	65.7 (0.2)	95.2 (0.2)	89.3 (0.2)	56.1 (0.6)	82.3 (0.1)	20.0 (0.2)	55.1 (0.2)	34.4 (0.3)
GPT-2 large	65.1 (0.3)	94.6 (0.4)	91.6 (0.2)	56.5 (0.5)	82.6 (0.1)	19.8 (0.1)	54.7 (0.1)	34.9 (0.3)
GPT-2 xlarge	72.1 (0.2)	94.2 (0.4)	89.5 (0.1)	**61.4 (0.7)**	**84.7 (0.2)**	24.2 (0.2)	59.0 (0.1)	**34.3 (0.3)**

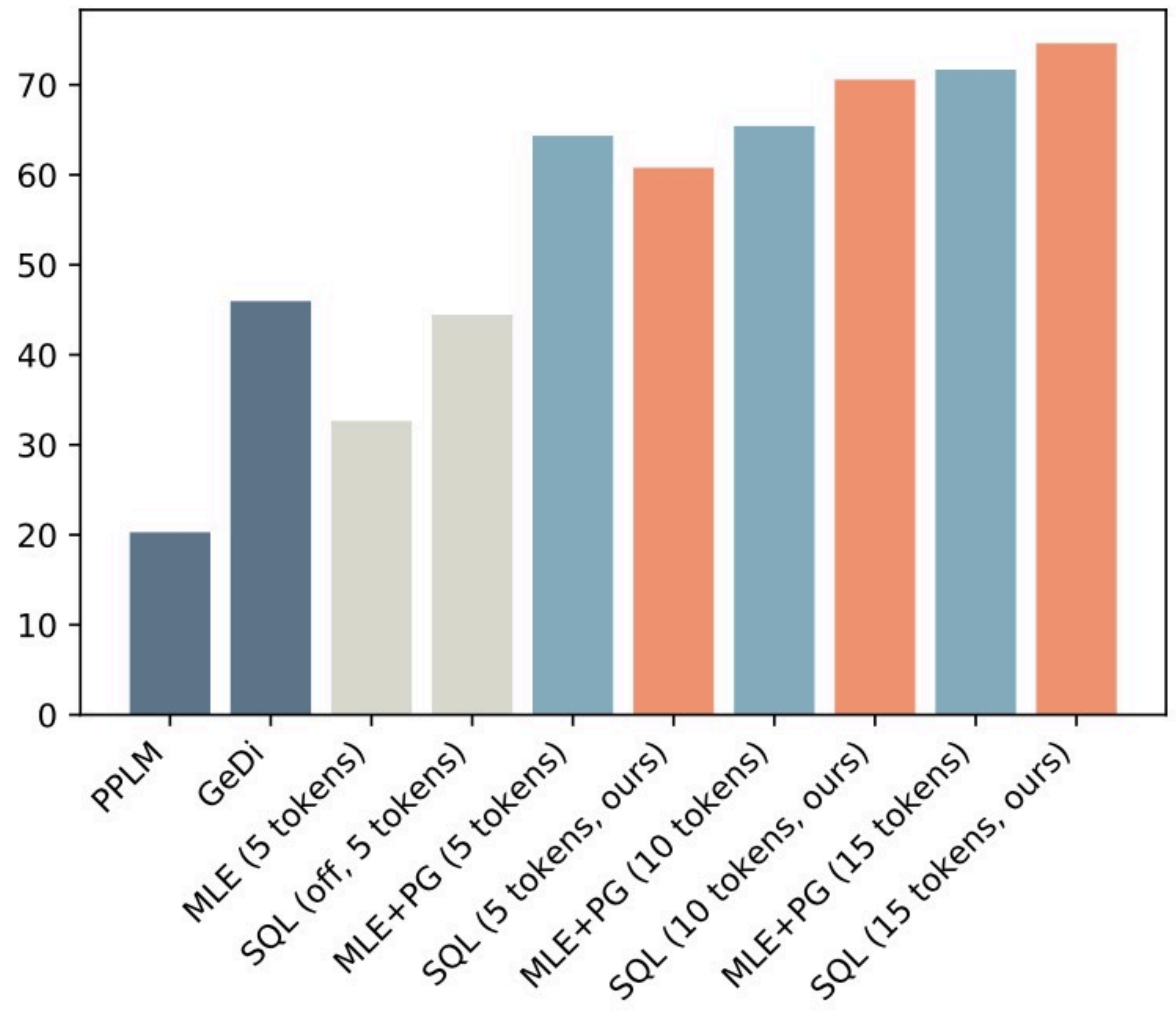
Table 4: Automatic evaluation of our method vs. baselines on the Yelp (Shen et al., 2017) sentiment transfer dataset.

Application (I): Prompt Optimization for Controlling LMs

- Topic-control generation



Application (I): Prompt Optimization for Controlling LMs



Topic accuracy

- Steered decoding: **PPLM, GeDi**
- **SQL** achieves better overall accuracy+fluency
- Prompt control by **SQL, MLE+PG** > **PPLM, GeDi**
  - and much faster at inference!

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

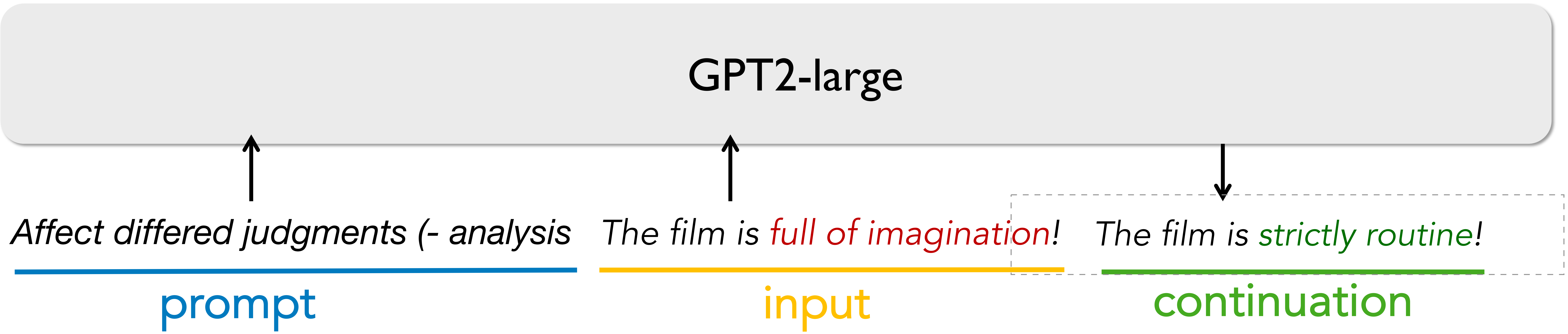
Application (I): Prompt Optimization for Controlling LMs

Interesting (Surprising) observations:

Application (I): Prompt Optimization for Controlling LMs

Interesting (Surprising) observations:

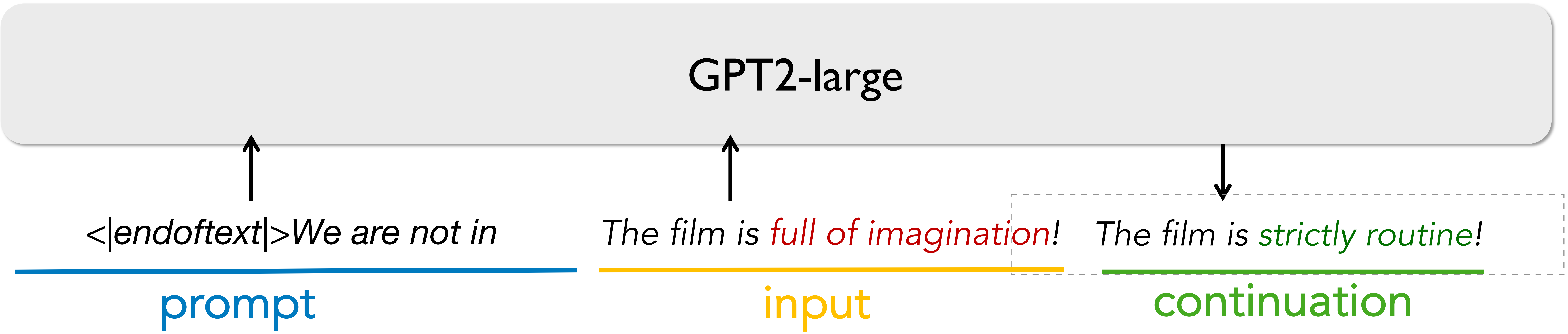
- Optimized prompts tend to be ungrammatical ***gibberish***



Application (I): Prompt Optimization for Controlling LMs

Interesting (Surprising) observations:

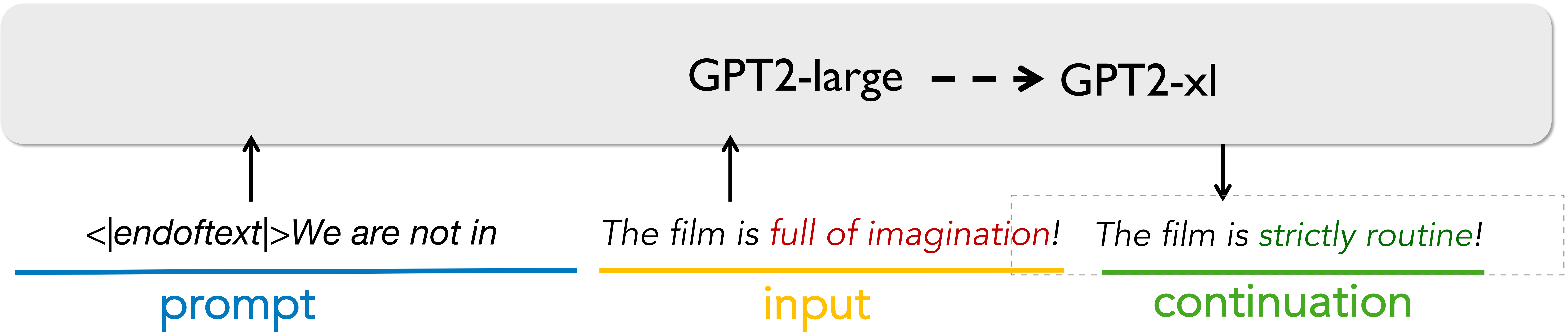
- Optimized prompts tend to be ungrammatical ***gibberish***
  - Adding fluency constraint harms the performance



Application (I): Prompt Optimization for Controlling LMs

Interesting (Surprising) observations:

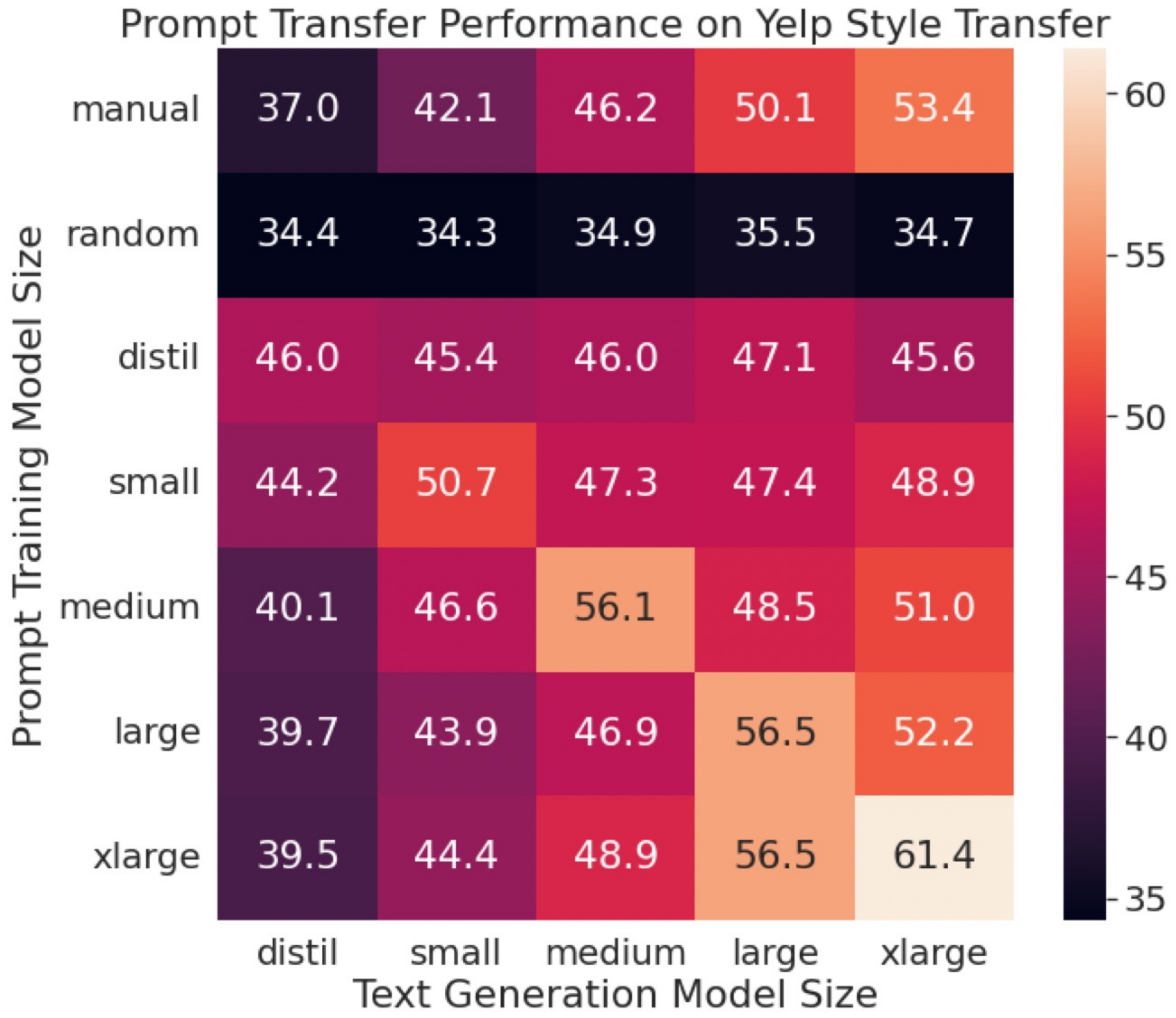
- Optimized prompts tend to be ungrammatical ***gibberish***
  - Adding fluency constraint harms the performance
- Those gibberish prompts are transferrable between LMs!



Application (I): Prompt Optimization for Controlling LMs

Interesting (Surprising) observations:

- Optimized prompts tend to be ungrammatical ***gibberish***
  - Adding fluency constraint harms the performance
- Those gibberish prompts are transferrable between LMs!



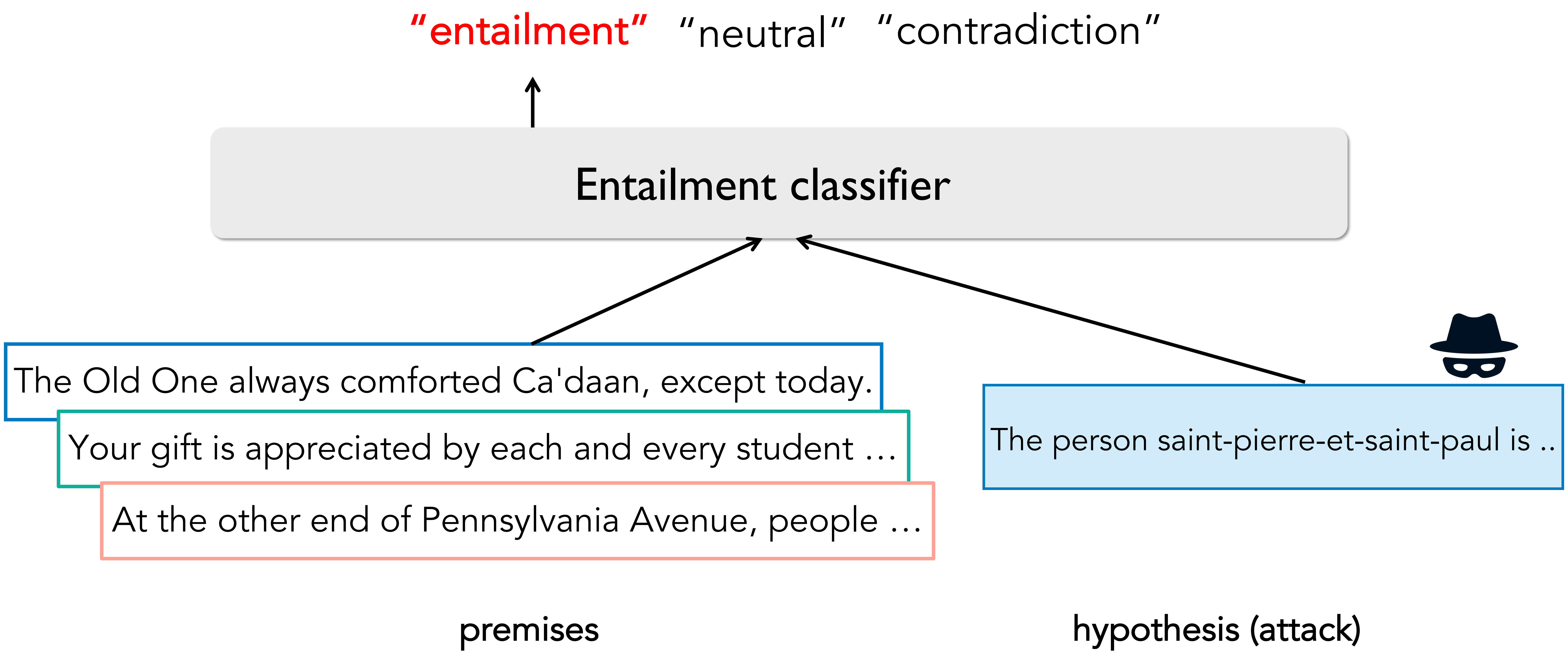
Application (I): Prompt Optimization for Controlling LMs

Interesting (Surprising) observations:

- Optimized prompts tend to be ungrammatical ***gibberish***
  - Adding fluency constraint harms the performance
- Those gibberish prompts are transferrable between LMs!

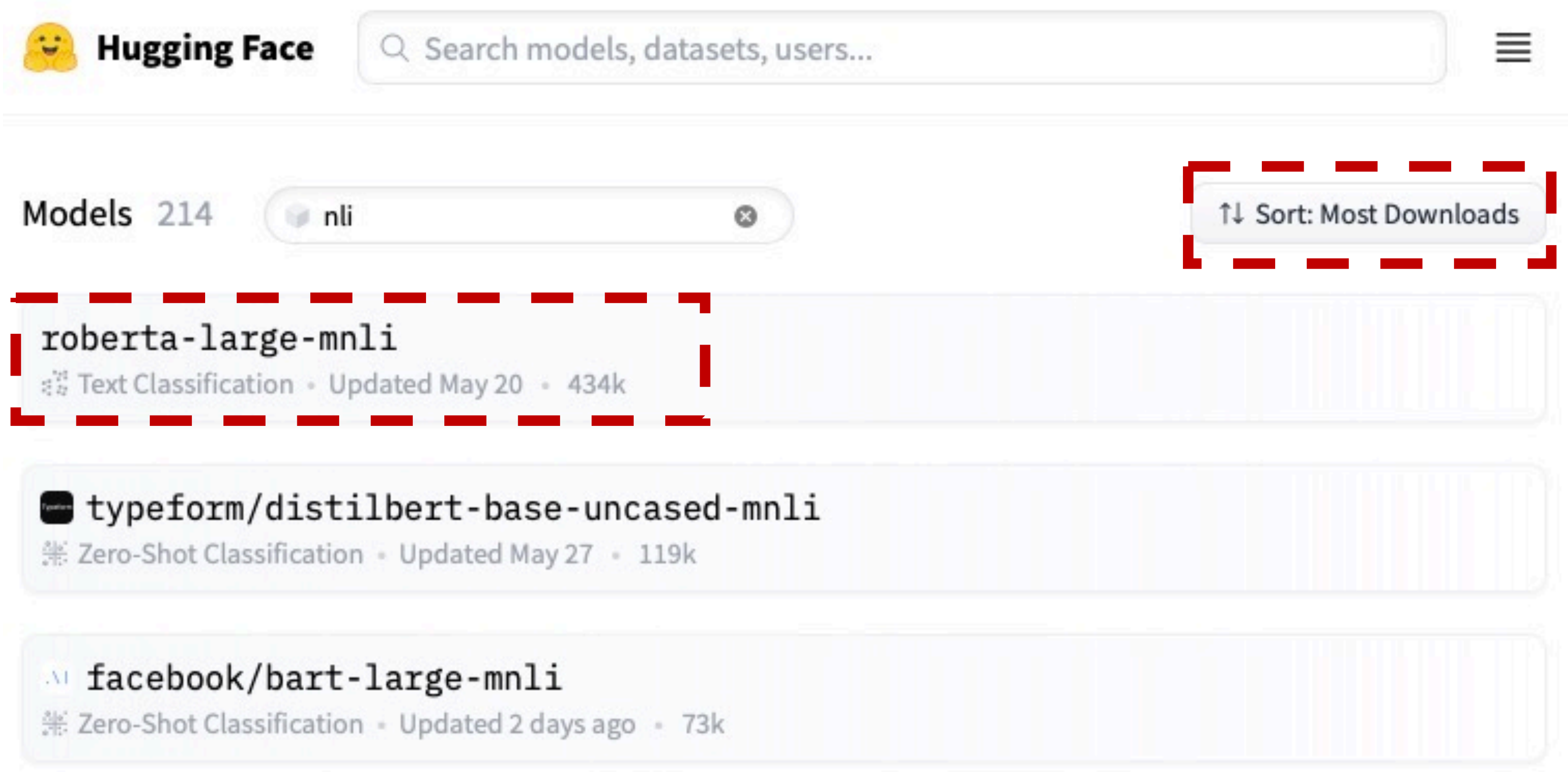
LM prompting may not follow human language patterns

Application (II): Universal Adversarial Attacks



Application (II): Universal Adversarial Attacks

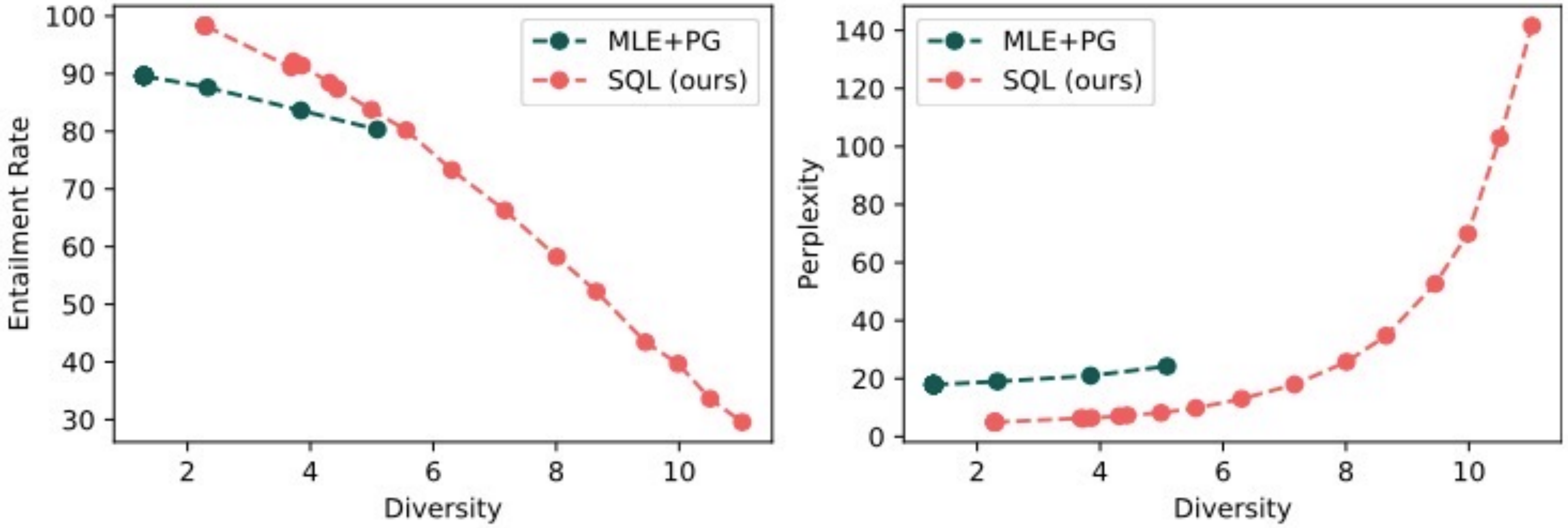
- Attacking entailment classifier
  - Generate **readable** hypotheses that are classified as "entailment" for **all** premises
  - *Unconditional* hypothesis generation model
- Training data:
  - No direct supervision data available
  - "Weak" data: all hypotheses in MultiNLI corpus
- Rewards:
  - Entailment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

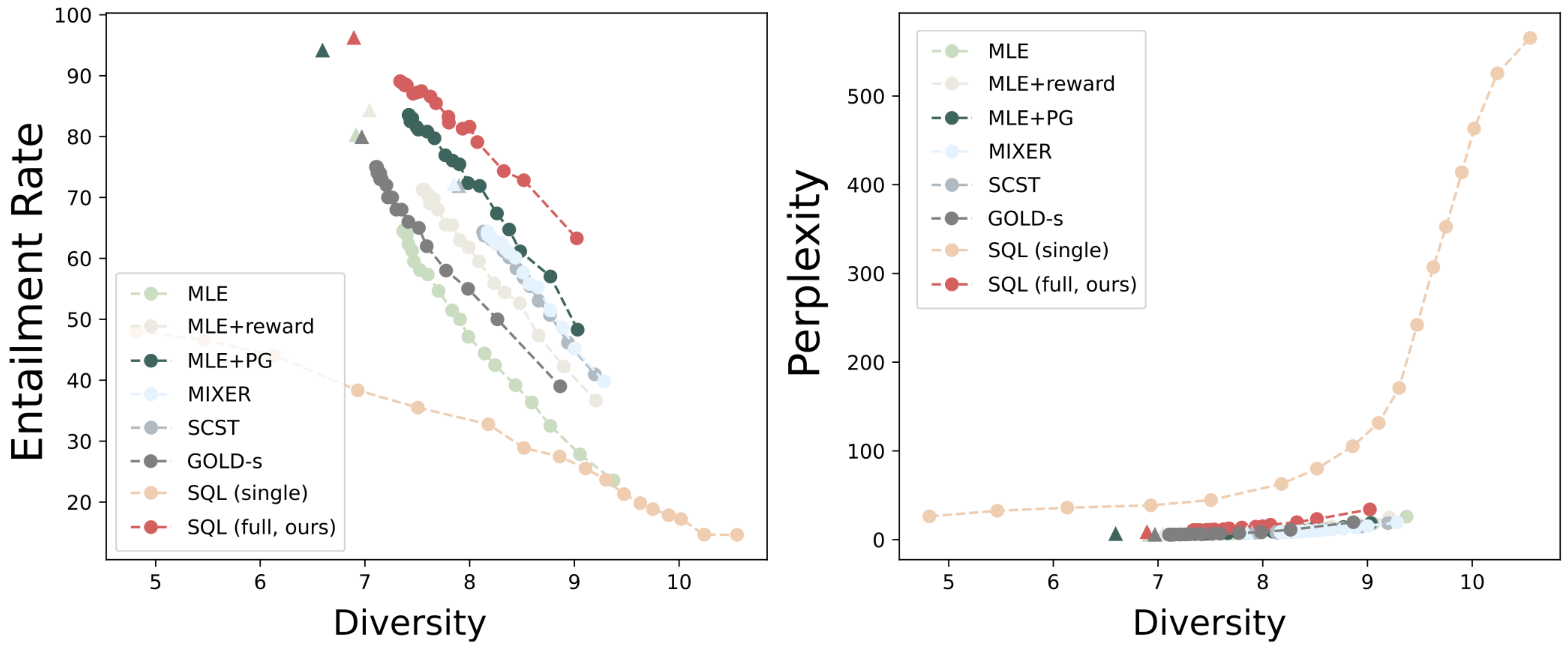
Samples of highest attack rate

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

- **MLE** (and variants) and pure off-policy RL (**GOLD-s**) do not work ← rely heavy on data quality
- **SQL (full)** > **MLE+PG** (PG alone does not work)



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

Key Takeaways

- Learning text generation from reward
- Previous RL for text generation (e.g., policy gradient, Q-learning):

Low data efficiency; unstable training; slow updates; sensitive to training data quality

- SQL
  - Objectives based on path consistency

  Stable training from scratch given sparse reward

  Fast updates given large action space
- Opens up enormous opportunities
  - For integrating more advanced RL (replay buffer, model-based RL, hindsight, …)
  - To enable massive new applications in text generation