Few-steps [Offline] Reinforcement Learning with Large Language Models

Antonio Mari
antonio.mari@epfl.ch

Matteo Santelmo
matteo.santelmo@epfl.ch

Stefano Viel
stefano.viel@epfl.ch

Abstract

We explore the application of offline reinforcement learning (RL) algorithms in few-step environments using large language models (LLMs). We target algorithms that optimize immediate reward. We employ two environments: the game of Wordle and a simpler word replacement task. Our experiments aim to assess how RL algorithms, specifically behavioral cloning (BC), filtered behavioral cloning, and our proposed reward-weighted behavioral cloning (Weighted-BC), perform in these settings. We find that Weighted-BC generalizes both BC and filtered BC, and our empirical results align with theoretical expectations. Additionally, we investigate the impact of performance conditioning on these models.

1 Introduction

Reinforcement Learning serves as a powerful tool for large language models training and there has been extensive research on Reinforcement Learning for Human Feedback (RLHF) (Ouyang et al. 2022), whose aim is aligning the output of a language model with human preferences. More generally, RL is useful to make LLMs accomplish tasks in a goal-directed manner.

However, most of the recent work on RL applied to LLMs has focused on "single-step" RL problems, where a single response is optimized with respect to some reward model. This is not surprising: it is likely much easier to evaluate improvements to algorithms for single-step text generation as compared to multi-step generation, with multi-turn dialogue requiring time-consuming studies with human participants.

In our work, we focus on understanding how some RL algorithms can be applied and extended in a more general multi-step settings. We are interested in few-steps environments to assess how algorithms suited for the single-step case (which optimize the immediate reward) perform on short horizon scenarios.

Our few-steps environment is Wordle (Wardle 2021), a game in which a player has to guess a secret five-letters word and has in total six guesses to do so. After every guess, the game provides feedback on which letters of the guess are present in the secret word and if they are correctly positioned. Moreover, we also experiment with a simple single-turn toy-environment, namely word replacement, which consists of replacing some specific words in a piece of text.

The reasons behind the choice of these environments: i) we have access to the exact reward, ii) we can generate a dataset with a given-policy, having access to an environment simulator. This way, we generate datasets with both expert policies and noisy policies. In the former case, we would like to clone the expert behavior, while in the latter case we are interested in learning the behavior that leads to high reward. To this end, we test which methods can well exploit good samples in low quality data.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

Contributions:

1. We formalize the Wordle environment in 2.1.2, explaining why it differs from the common settings where RL is usually applied on LLMs.
2. We analyze an algorithm proposed by our laboratory supervisors, namely the *weighted Behavioral Cloning*, understanding how its formulation changes under this different type of environment. We detail all the subtleties of this approach, hoping to shed lights on its potential pitfalls.
3. We show how Behavioral Cloning and Filtered Behavioral cloning represent limit cases of Weighted-BC, as its hyperparameter β tends to infinity or to zero, respectively.
4. We experiment these methods on different quality datasets, to see if any real benefit we can get from weighted-BC.

2 Methods

In this section, we formalize the environments used in our work, then present the RL algorithms used to fine-tune our LLMs. As pre-trained language models, we used Pythia-14m (Biderman et al. 2023) for the word replacement task and OPT-350M (S. Zhang et al. 2022) for Wordle.

2.1 Environments

We experiment with a simple single-step environment, namely "word replacement", in order to perform quick tests of our method, but the main focus is the Wordle environment. That is why we formalize Wordle in detail and we define the reward-weighted behavioral cloning considering the underlying Markov decision process. Both environments provide explicit and easily computable reward and we have access to a simulator to run online evaluation of fine-tuned LLMs.

2.1.1 Word Replacement

Word replacement is a single-turn toy environment where the player's task consists of replacing some words in an input sentence, following a replacement map learned during training. Specifically, given a vocabulary $\mathcal{V}$ and a replacement map $d : \mathcal{K} \to \mathcal{T}$ where $\mathcal{K}, \mathcal{T} \subseteq \mathcal{V}$, the model task is simply taking an input sentence and outputting the same sentence where all occurrences of a word $k \in \mathcal{K}$ are replaced with the corresponding target word $t = d(k)$. A reward between between 0 and 8 was assigned to a response proportionally to the number of words correctly replaced, and set to -8 if the model was replacing a word that shouldn't have been replaced, in order to discourage illegal substitutions.

2.1.2 Wordle

In this section, we formalize the Markov Decision Process (MDP) of the Wordle game. We denote as Σ the lower-case English letters alphabet, (i.e., $\Sigma = \{a, b, \dots, z\}$) and we choose a vocabulary $V \subset \Sigma^5$, where, for a set A, $A^n = A \times A \times \cdots \times A$ indicates the Cartesian product of A with himself n-times. Note that V contains only English five-letter words, and in the original game $|V| = 2309$ (while for our simplified experiments we choose a smaller dictionary, $|V| = 100$). In every game, a random "secret" word is chosen uniformly form the vocabulary, i.e. $w \sim \mathcal{U}(V)$ and represents the word to guess. We also define the feedback set $\mathcal{F} = \{\langle - \rangle, \langle y \rangle, \langle g \rangle\}^5$, where a $\langle - \rangle$ represents a letter that is not present in the secret word w, $\langle y \rangle$ represents a letter that is present in w but in another position, $\langle g \rangle$ indicates that the letter is correctly positioned as in w. So the feedback is deterministic when w is fixed. In general, w is not known so the feedback will be a random function.

The Wordle game MDP is a tuple $(\mathcal{S}, \mathcal{A}, r, P, \mu, \gamma)$, where

1. A *state* $s = (g_1, f_w(g_1), g_2, f_w(g_2), \dots, g_6, f_w(g_6)) \in (\Sigma^5 \times \mathcal{F})^6 =: \mathcal{S}$ represents a valid board.
   - $g_i \in V \cup \{\perp\}, \forall i \in \{1, 2, \dots, 6\}$, i.e. each guess on the board is a five-letters English word or is empty. The latter case is represented by the special symbol $\perp$.
   - $f_w(g_i) =\in \mathcal{F} \cup \{\perp\}$, i.e. each feedback follows a guess and depends on the secret word, which is not observed. Note than $f_w(g) = \perp \iff g = \perp, \forall w \in V$

Note that not every board states is valid. Informally, after an empty guess $\perp$ there could not be any non-empty guess on the board and the feedback for each guess has to adhere with the secret word w. With that said, we denote the set of all the valid boards given the secret word w as $\mathcal{D}(w)$.

2. An *action* $a = g \in V$ is a valid guess, so the action space $\mathcal{A} = V$ is discrete and any guess can be performed at any state.
3. The *reward* $r_w(s, a)\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ depends on the secret word w and is defined as $r_w(s, a) = h(f_w(a))$. In our case, $h = 0 \cdot \#(\langle g \rangle) - 3 \cdot \#(\langle y \rangle) - 5 \cdot \#(\langle - \rangle)$ counts the occurrences of each symbol in the feedback and assign them a score (0 for $\langle g \rangle$, -3 for $\langle y \rangle$, -5 for $\langle - \rangle$). This way, the range of the reward is $[-25, 0]$, with the lowest reward achievable when the feedback is $(\langle - \rangle, \langle - \rangle, \langle - \rangle, \langle - \rangle, \langle - \rangle)$ and the highest corresponding to the correct guess $(\langle g \rangle, \langle g \rangle, \langle g \rangle, \langle g \rangle, \langle g \rangle)$.
4. The transition probabilities $P(s'|s, a)$ are characterized as follows:

$$P(s'|s, a) = \sum_{w \in V} p(w, s'|s, a) = \sum_{w \in V} p(w) p(s'|s, a, w) = \frac{1}{|V|} \sum_{w \in V} p(s'|s, a, w)$$

Note that $p(s'|s, a, w)$ is actually deterministic because an action uniquely determines the next guess present on the board and the feedback is a deterministic function of w. Moreover, if we define s and $s|(a, w)$ to be of the form

$$\begin{aligned} s &= (g_1, f_w(g_1), \dots, g_i, f_w(g_i), \perp, \perp, \dots) \\ s|(a, w) &:= (g_1, f_w(g_1), \dots, g_i, f_w(g_i), a, f_w(a), \dots) \end{aligned}$$

when the action a and the secret word w are fixed, then for all $s \in \mathcal{S}$:

$$P(s'|s, a, w) = \begin{cases} 1 & \text{if } s' = s|(a, w), \\ 0 & \text{otherwise} \end{cases}$$

Actually, the transition probability $P(s'|s, a)$ is a mixture of deterministic discrete distributions where all components are weighted uniformly and the stochasticity of the transition is directly linked to the stochasticity of the feedback, which depends on the latent variable w. Also note that the transition probabilities are actually independent of the state s (that is, all the previous guesses and feedback received) and really depend only on the action and the secret word. With that said, we have to include all this information because the policy learned to solve the game *has to* use it to make new guesses.

5. The *initial distribution* μ is deterministic since the starting state for every game is $(\perp, \perp, \dots)$.
6. Since we are dealing with a finite horizon environment, the discount factor $\gamma = 1$ (note: we will change this setting in section 2.4)

In our work, the policy $\pi(\cdot|s)$ is parameterized by a language model and we encode a state s in a prompt-string. Given a state s:

$$\begin{aligned} s =& ((a, r, i, s, e), (\langle - \rangle, \langle y \rangle, \langle - \rangle, \langle y \rangle, \langle y \rangle), \\ & (r, o, u, t, e), (\langle g \rangle, \langle - \rangle, \langle y \rangle, \langle - \rangle, \langle y \rangle), \\ & (r, u, l, e, s), (\langle g \rangle, \langle y \rangle, \langle - \rangle, \langle y \rangle, \langle g \rangle), \\ & \perp, \perp, \perp, \perp, \perp, \perp) \end{aligned}$$

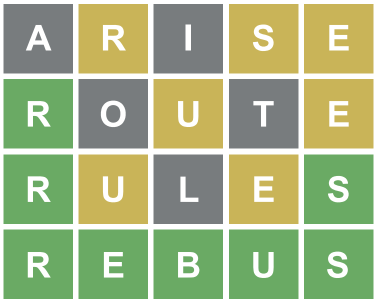
Its string encoding is reported in figure 1.

2.2 Behavioral Cloning

A commonly used offline RL approach for language model training is *behavioral cloning* (BC), which consists of performing supervised fine-tuning on a dataset $\mathcal{D}$ composed of (s, a) pairs where $a \sim \pi_E(s)$ and π_E is an expert policy which we seek to clone. The maximum likelihood objective:

$$\pi_{\text{MLE}} = \text{argmax}_\pi \sum_{(s,a) \in \mathcal{D}} \log \pi(a|s)$$

A common variation of BC is the *filtered behavioral cloning*, which first filters the data following some criteria, in order to learn only from the most successful samples, rejecting the others. In our case, a natural criterion for filtering is the reward r.




```
Guess a five-letter word
based on previous attempts and feedback.
Feedback symbols: '-' (letter not in word),
'y' (letter in word but wrong position),
'g' (letter correct and in right position).
Review previous guesses and feedback:
Guess 1: a r i s e | Feedback: - y - y y
Guess 2: r o u t e | Feedback: g - y - y
Guess 3: r u l e s | Feedback: g y - y g
### Response:
```

Figure 1: An example of a prompt for a Wordle game guess. Note that letters are separated by white spaces to make sure that they are all tokenized separately. The expected correct guess is "r e b u s".

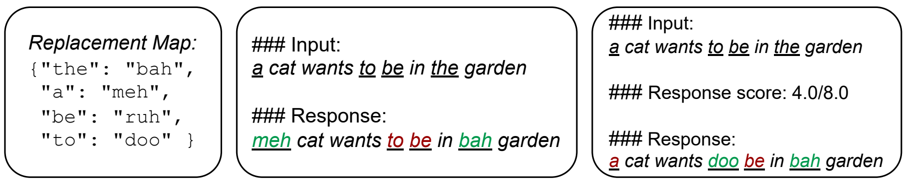


Figure 2: Example of data sample for word replacement task, generated using the replacement map on the left and probability of replacement $p = 0.5$, therefore providing a moderately noisy signal to the model. In the center and on the right respectively, two samples without and with reward conditioning.

2.3 Performance conditioning Behavioral Cloning

A surprisingly effective method to leverage the generalization and in-context learning abilities of LLMs is *performance conditioned behavioral cloning* (Shypula et al. 2023). Inspired by prompting strategies (T. Zhang et al. 2023) and offline RL (L. Chen et al. 2021), this method adapts BC for LLMs by directly introducing a reward signal into the training process by explicitly incorporating this information in the prompt. In practice, this means modifying each (s, a, r) triple to a new one $(\tilde{s}, a, r)$, where $\tilde{s}$ encodes both information about the current state and the reward $r = r(s, a)$. This allows the model to implicitly learn a mapping between responses and their correctness, which can be used during inference to improve results by asking the model to provide a response with the best reward.

2.4 Reward-Weighted Behavioral Cloning

Our main contributions revolve around the *Reward-Weighted Behavioral Cloning* (**Weighted-BC**). In this section, we introduce its formulation and analyze the approximation involved, identifying where potential pitfalls may reside. Moreover, we show how this approach generalizes both BC and Filtered-BC.

Reward optimization as a limit Given the Wordle environment introduced in section 2.1.2 and for $\gamma \in (0, 1)$, the value function of a policy π is defined as follows:

$$V^\pi(s) := \mathbb{E}\left[\sum_{t=1}^{6} \gamma^{t-1} r_w(s_t, a_t) | s_1 = s, \pi\right]$$

As we take the limit as $\gamma \to 0$, the horizon degenerates and only the immediate reward is considered:

$$\lim_{\gamma \to 0} V^\pi(s) = \mathbb{E}_{w \sim \mathcal{U}(V), s \sim \mathcal{D}(w), a \sim \pi(\cdot|s)} \left[r_w(s, a)\right]$$

Note that for a single-step environments and $\gamma = 1$ this limit matches $V^\pi(s)$. Conversely, in a general multi-step environment, if we approximate the value function with this limit then the quality of our approximation is going to be worse as the number of steps grows. Our working assumption: in few-steps environment (such as Wordle, where the number of steps $T = 6$), we assume that such approximation is "good enough", in the sense that by optimizing the immediate reward we could good results as if we optimized the quality/value function.

Starting point: DPO loss The loss commonly used for RLHF is the following (cfr DPO paper Rafailov et al. 2023, we adapted the expected value on Wordle data):

$$\max_{\theta} \mathbb{E}_{w\sim\mathcal{U}(V), s\sim\mathcal{D}(w), a\sim\pi_\theta(\cdot|s)}[r_w(s,a)] - \beta D_{KL}[\pi_\theta(\cdot|s)||p(\cdot|s)]$$

where p is a reference policy (a pre-trained LLM or an handcrafted policy), defined over states s, and $\beta \geq 0$ is a regularization parameter that modulates how far we may stray from $p(\cdot|s)$.

The expected value is taken with respect to the space of the observable data, which represents all the possible wordle games. A sample from this dataset space is drawn first by sampling the secret word uniformly from the vocabulary $w \sim \mathcal{U}(V)$ and then sampling a "random" state $s \sim \mathcal{D}(w)$, where $\mathcal{D}(w)$ is the set of legal board states which are consistent with the secret word w. We can assume that a sample is draw uniformly random from this set, but this assumption is actually violated in the datasets that we are going to use, where full trajectories are drawn (see section 3.2).

As shown in Rafailov et al. 2023 (and here adapted due to the introduction of the expected value on w), the analytical solution to this objective is:

$$\pi^\star(a|s,w) = \frac{1}{Z(s,w)} p(a|s) \exp\left(\frac{1}{\beta} r_w(s,a)\right)$$

where $Z(s,w)$ is the (intractable) partition function that ensures that $\sum_{a\in\mathcal{A}} \pi^\star(a|s,w) = 1$.

Nota Bene: $\pi^\star$ is conditioned on both s and w, but the policy that we are interested to clone should not be conditioned on w. A natural way to obtain it is marginalizing out w (overloading $\pi^\star$ symbol):

$$\pi^\star(a|s) = \sum_{w\in V} p(w|s) \cdot \pi^\star(a|s,w)$$

Another *crucial approximation* that we do next is assuming $p(w|s) = p(w)$, i.e. w is independent of s, so that:

$$\mathbb{E}_{w\sim\mathcal{U}(V)}[\pi^\star(a|s,w)] = \sum_{w\in V} p(w)\pi^\star(a|s,w) = \pi^\star(a|s)$$

Weighted-BC Objective *Proposed idea*[1]*: optimize a cross-entropy objective whose solution is* $\pi^\star(\cdot|s)$ *(under the approximation as before)*

$$\begin{aligned}
\mathcal{L}(\theta) &= \mathbb{E}_{w\sim\mathcal{U}(V), s\sim\mathcal{D}(w)}\left[\sum_{a\in\mathcal{A}} \pi^*(a|s,w)\log\pi_\theta(a|s)\right] \\
&= \mathbb{E}_{w\sim\mathcal{U}(V), s\sim\mathcal{D}(w)}\left[\frac{\sum_{a\in\mathcal{A}} p(a|s)\exp\left(\frac{1}{\beta}r_w(s,a)\right)\log\pi_\theta(a|s)}{\sum_{a\in\mathcal{A}} p(a|s)\exp\left(\frac{1}{\beta}r_w(s,a)\right)}\right] \\
&= \mathbb{E}_{w\sim\mathcal{U}(V), s\sim\mathcal{D}(w), a\sim p(\cdot|s)}\left[\frac{1}{Z(s,w)}\exp\left(\frac{1}{\beta}r_w(s,a)\right)\log\pi_\theta(a|s)\right]
\end{aligned}$$

Since the explicit value of $p(\cdot|s)$ is not directly accessible, for each state s we sample n times from this distribution and denote the obtained set of samples as A_s, with $|A_s| = n$. Consequently, we set $\hat{p}(\cdot|s) := \mathcal{U}(A_s)$, so we can approximate both the expected value over $p(\cdot|s)$ and $Z(s,w)$ using $\hat{p}(\cdot|s)$ instead. Our sampled loss is denoted as $\hat{\mathcal{L}}$:

$$\mathcal{L}(\theta) \approx \hat{\mathcal{L}}(\theta) := \mathbb{E}_{w\sim\mathcal{U}(V), s\sim\mathcal{D}(w), a\sim\hat{p}(\cdot|s)}\left[\frac{\exp\left(\frac{1}{\beta}r_w(s,a)\right)}{\sum_{a\in A_s}\exp\left(\frac{1}{\beta}r_w(s,a)\right)}\log\pi_\theta(a|s)\right]$$

Some considerations:

- The objective is a likelihood where actions are weighted using their reward. The population population weights are, for all $w \in V$, $s \in \mathcal{D}(w)$ and $a \in \mathcal{A}$:

$$\rho(a|s,w) = \frac{\exp\left(\frac{1}{\beta}r_w(s,a)\right)}{\sum_{a\in\mathcal{A}}\exp\left(\frac{1}{\beta}r_w(s,a)\right)}$$

[1] Explain exactly who proposed it and how

- These weights are approximated using sets of samples A_s:

$$\hat{\rho}(a|s,w) = \frac{\exp\left(\frac{1}{\beta} r_w(s,a)\right)}{\sum_{a \in A_s} \exp\left(\frac{1}{\beta} r_w(s,a)\right)}$$

- The sampled loss $\hat{\mathcal{L}}(\theta)$ is a biased estimator of $\mathcal{L}(\theta)$ because the expected value does not distribute over products and divisions (unless independence holds, which is not the case).

Weighted-BC generalizes BC and Filtered-BC Let us analyze the limit behavior of the weighted-BC. Let $n = |A_s|$ for every $s \in \mathcal{D}$ (where $\mathcal{D}$ is a generic dataset) and let b be the batch size used for our updates.

1. When $\beta \to 0$, the weights $\rho(a|s,w) \to 1$ on the action with maximum reward (assuming it is unique). It corresponds to Filtered-BC, where the batch size is b/n.
2. When $\beta \to \infty$, $\rho(a|s,w) = 1/n$, $\forall a \in A_s$. It corresponds to BC, since the constant $1/n$ does not affect the solution of the optimization problem.

3 Experiments

In this section, we evaluate the mentioned methods on both environments. In our experiments, the main research questions are:

Q1. Does weighted-BC empirically match our theoretical expectation? How the hyper-parameter β affects the performance? Can weighted-BC outperform the baselines?

Q2. Does the performance-conditioning help in solving the tasks?

Q3. What changes if we have a dirty dataset, which comes from a policy with sub-optimal behavior?

3.1 Word Replacement

Given the simplicity of this task, the experiments were conducted on a less powerful language model and using a quite noisy signal. In particular, the performance of different models discussed in section 2 were compared using a dataset generated with a probability of replacement of the key-words of $p = 0.25$. The results, in figure 3, show that despite having a relatively noisy signal, filtered BC proved to always achieve better rewards, both in the performance conditioned and unconditioned cases. A possible explanation behind this unexpected behavior stands in the way the reward is defined, in particular because in all cases where there is no word to be replaced in the input, the non-filtered methods will see more samples being identical while not being useful for the model to learn the task, despite their high reward. 3.

3.2 Wordle

We test our wordle environment using two datasets, the first one generated using an handcrafted expert policy π_E and the second generated using a noisy policy, which consists of a mixture between π_E and a random policy.

The datasets are generated in the following way:

1. A secret word w is sampled from $\mathcal{U}(V)$. Then, a Wordle game with that secret word is played by drawing samples from $p(\cdot|s)$. At each turn of a game, N samples are generated and they are gathered in the set A_s, along with their rewards. Since multiple actions are generated each turn, a random one is then kept for next turn and the state s is updated.
2. This process is repeated M times, so that at the end we will have collected trajectories that correspond to M Wordle games.

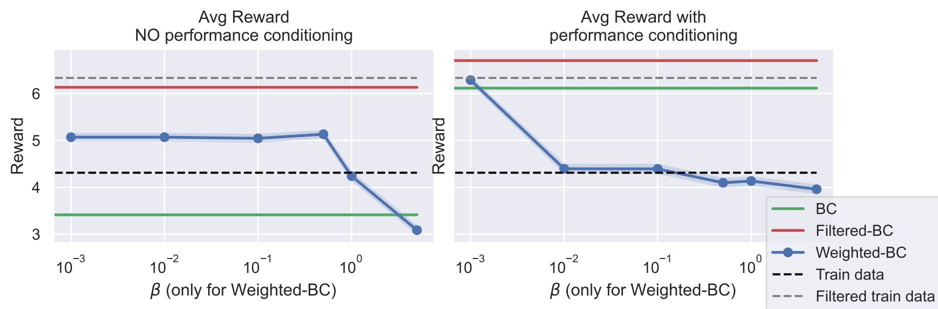


Figure 3: Average reward (with 0.95 confidence interval) for word replacement single-turn task. Weighted BC was tested over different βs and both BC and Weighted BC were trained with suboptimal supervisions (dashed line), while Filtered BC's reference dataset had more quality samples. Weigthed BC's performance are between BC and Filtered BC, as theoretically expected, approaching Filtered BC and standard respectively with small β and high β.

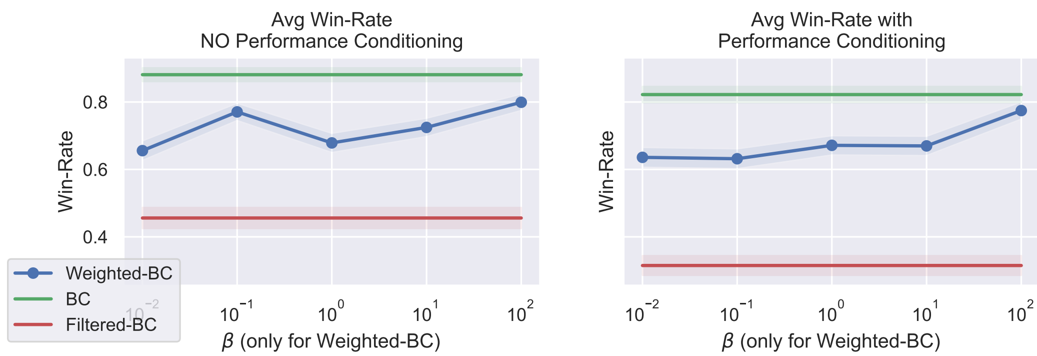


Figure 4: Win-rate (over 1000 games) of models trained on the **expert policy dataset**. **5000 games were used for training**, where at every step 5 independent guesses are sampled from the expert policy. Then, **the guess associated with the highest reward is kept for Filtered-BC** while a **random guess is used to continue the game**. The batch size is 128 for BC and Weighted-BC, 25 for Filtered-BC, and the learning rate is consistently 5e-05. The same number of gradient updates was performed across models, ensuring the convergence of the training.

3.2.1 Expert policy dataset

The expert policy is handcrafted and at each step filters all the words in the vocabulary based on the known constraints and randomly samples among the filtered words. The win-rate of this policy over 5000 games is 99.84%.

The results are showed in fig. 4, and the answers to our research questions:

A1. The empirical results match the expectation. Discarding data is harmful because all guesses are generated by the expert policy π_E and, despite some fluctuations, by modulating β it appears that we approach both limits (i.e., Filtered-BC for small β, BC for high β), closing the gap between the two methods.

A2. Performance-conditioning leads to worse win-rates in both baselines, while it does not change much for weighted-BC. At this stage, it is not really clear if these results are statistically significant, so that runs with multiple seeds would help to assess whether there is a real difference between the two plots of fig. 4.

3.2.2 Mixture of Expert and Random Policy dataset

The "mixture policy" is a mixture between the expert policy and a random policy, with the mixture weight $p = 0.4$. This means that, with 40% probability, a sample is drawn from a random policy (so

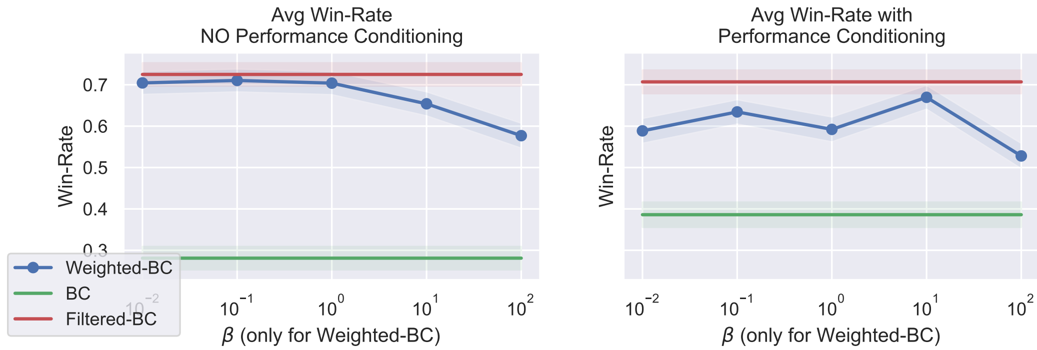


Figure 5: Win-rate of models trained on the **mixture policy dataset**. Same training settings as in fig. 4

from a uniform distribution over the whole dictionary) and with 60% probability it is drawn from π_E. The win-rate of the mixture policy over 5000 games is 92.4%. This good performance is explained by the fact that sampling random words helps exploration of new constraints, so subsequent non-random guesses will still leverage the constraints discovered. Results on this dataset are reported in fig. 5

A1. Also in this case, empirical results match the expectation. Opposite to the previous case, Filtered-BC dominates by only retaining the best examples (1/5 of them) and we can clearly observe weighted BC approaching filtered-BC for small β.

A2. Similar considerations than before can be drawn. The only difference is that we observe a boost in BC win-rate, but we are far from being able to reach any conclusion on performance conditioning.

A3. Generally, in the expert policy dataset we can reach higher win-rates (> 80%) and, according to our intuition, leveraging all the available high-quality data, as well as filtering only the good-data in a dirty dataset, is the strategy that achieves the highest win-rate.

4 Discussion

As claimed in DPO paper, the hyper-parameter β regulates the "strength" of the KL regularizer term. This intuition is consistent with our experiments on weighted-BC. In particular, high β means strong adherence to the reference policy. This will lead to better results if the data was generated using an expert policy. Conversely, for small $\beta \approx 0$, we optimize the reward without adhering to the reference policy, thing that is desirable when the data is dirty or noisy.

In the high quality data scenario, as expected, the best results are obtained by the methods that are leveraging more data, i.e. BC and Weighted BC with high β, while for poor quality data the best results are achieved by methods which leverage samples with higher rewards, either by filtering or by using a very accentuated weighting.

It is worth noting that if we do not have prior information about the cleanliness of our data then weighted-BC seems to be a proper choice to achieve reasonable performance in different scenarios.

Future works should firstly address a more exhaustive experimentation to confirm the validity of our results and possibly compare with more complex algorithms capable of optimizing over the cumulative reward or an approximation of it, going over the limitation of optimizing on the immediate reward.

References

Biderman, Stella et al. (2023). “Pythia: A suite for analyzing large language models across training and scaling”. In: *International Conference on Machine Learning*. PMLR, pp. 2397–2430.
Chen, Lili et al. (2021). “Decision transformer: Reinforcement learning via sequence modeling”. In: *Advances in neural information processing systems* 34, pp. 15084–15097.

Ouyang, Long et al. (2022). "Training language models to follow instructions with human feedback". In: *Advances in neural information processing systems* 35, pp. 27730–27744.
Rafailov, Rafael et al. (2023). *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. arXiv: 2305.18290 [cs.LG].
Shypula, Alexander et al. (2023). "Learning performance-improving code edits". In: *arXiv preprint arXiv:2302.07867*.
Wardle, Josh (2021). *Wordle*. https://www.nytimes.com/games/wordle/index.html.
Zhang, Susan et al. (2022). "Opt: Open pre-trained transformer language models". In: *arXiv preprint arXiv:2205.01068*.
Zhang, Tianjun et al. (2023). "The wisdom of hindsight makes language models better instruction followers". In: *International Conference on Machine Learning*. PMLR, pp. 41414–41428.