score \( s_a \in [0.1, 1] \) is assigned to arm \( a \in [K] \), and the probability of action \( a \) is

\[
\mu(a|x) = \frac{0.3 \times s_a}{\sum_{a'} s_{a'}} + 0.7 \times I(a \in c).
\]

This kind of sampling ensures two useful properties. First, every action has a non-zero probability, so such a dataset suffices to provide an unbiased offline evaluation of any policy. Second, actions corresponding to correct labels have higher observation probabilities, emulating the typical setting where a baseline system already has a good understanding of which actions are likely best.

We now consider the two tasks of evaluating a static policy and an adaptive policy. The first serves as a sanity check to see how well the evaluator works in the degenerate case of static policies. In each task, a one-vs-all reduction is used to induce a multi-class classifier from either fully or partially labeled data. In fully labeled data, each example is included in data sets of all base binary classifiers. In partially labeled data, an example is included only in the data set corresponding to the action chosen. We use the LIBLINEAR [9] implementation of logistic regression. Given a classifier that predicts the most likely label \( a^* \), our policy follows an \( \varepsilon \)-greedy strategy with \( \varepsilon \) fixed to 0.1; that is, with probability 0.9 it chooses \( a^* \), otherwise a random label \( a \in [K] \). The reward estimator \( \hat{r} \) is directly obtained from the probabilistic output of LIBLINEAR (using the “-b 1” option). The scaling parameter is fixed to the default value 1 (namely, “-c 1”).

**Static Policy Evaluation.** In this task, we first chose a random 10\% of \( D \) and trained a policy \( \pi_0 \) on this fully labeled data. From the remainder, we picked a random “evaluation set” containing 50\% of \( D \). The average loss of \( \pi_0 \) on the evaluation set served as the ground truth. A partially labeled version of the evaluation set was generated by the conversion described above; call the resulting dataset \( D' \). Finally, various offline evaluators of \( \pi_0 \) were compared against each other on \( D' \).

**Adaptive Policy Evaluation.** In this task, we wanted to evaluate the average online loss of the following adaptive policy \( \pi \). The policy is initialized as a specific “offline” policy calculated on random 400 fully observed examples (1\% of \( D \)). Then the “online” partial-feedback phase starts (the one which we are interested in evaluating). We update the policy after every 15 examples, until 300 examples are observed. On policy update, we simply use an enlarged training set containing the initial 400 fully labeled

---

1To avoid risks of overfitting, for evaluators that estimate \( \hat{r} \) (all except RS), we split \( D' \) into two equal halves, one for training \( \hat{r} \), the other for running the evaluator. The same approach was taken in the adaptive policy evaluation task.

and additional partially labeled examples. The offline training set was fixed in all trials. The remaining data was split into two portions, the first containing a random 80\% of \( D \) for evaluation, the second containing 19\% of \( D \) to determine the ground truth. The evaluation set was randomly permuted and then transformed into a partially labeled set \( D' \) on which evaluators were compared. The generation of \( D' \) was repeated in 50 trials, from which bias and standard deviation of each evaluator were obtained. To estimate the ground-truth value of \( \pi \), we simulated \( \pi \) on the randomly shuffled (fully labeled) 19\% of ground-truth data 2000 times to compute its average online loss.

**Compared Evaluators.** We compared the following evaluators described earlier: DM for direct method, RS for the unbiased evaluator in [19] combined with rejection sampling, and DR-ns as in Algorithm 1 (with \( c_{\text{max}} = 1 \)). We also tested a variant of DR-ns, which does not monitor the quantile, but instead uses \( c_t \) equal to \( \min_D \mu(a|x) \); we call it WC since it uses the worst-case (most conservative) value of \( c_t \) that ensures unbiasedness of rejection sampling.

**Results.** Tables 1 and 2 summarize the accuracy of different evaluators in the two tasks, including \( \text{rmse} \) (root mean squared error), bias (the absolute difference between evaluation mean and the ground truth), and \( \text{stddev} \) (standard deviation of the evaluation results). It should be noted that, given the relatively small number of trials, the measurement of bias is not statistically significant. So for instance, it cannot be inferred in a statistically significant way from Table 1 that WC enjoys a lower bias than RS. However, the tables provide 95\% confidence interval for the \( \text{rmse} \) that allows a meaningful comparison.

It is clear from both tables that although rejection sampling is guaranteed to be unbiased, its variance usually is the dominating source of \( \text{rmse} \). At the other extreme is the direct method, which has the smallest variance but often suffers high bias. In contrast, our method DR-ns is able to find a good balance between these extremes and, with proper choice of \( q \), is able to yield much more accurate evaluation results. Furthermore, compared to the unbiased variant WC, DR-ns’s bias appears to be modest.

It is also clear that the main benefit of DR-ns is its low variance, which stems from the adaptive choice of \( c_t \) values. By slightly violating the unbiasedness guarantee, it increases the effective data size significantly, hence reducing the variance of its evaluation. In particular, in the first task of evaluating a static policy, rejection sampling was able to use only 264 examples (out of the 20K data in \( D' \)) since the minimum value of \( \mu(a|x) \) in the exploration data was very small; in contrast, DR-ns was able to use 523, 3375, 4279, and 4375 examples for \( q \in \{0, 0.01, 0.05, 0.1\} \), respectively. Similarly, in the adaptive policy evaluation task, with DR-ns\((q > 0)\), we could extract many more online tra-