A SUPPLEMENTARY MATERIAL

A.1 MAPO

Liang et al. (2018) make the important observation that the expected return objective can be expressed as a sum of two terms: a summation over the trajectories inside a context specific buffer $\mathcal{B}^+(\boldsymbol{x})$ and a separate expectation over the trajectories outside of the buffer:

$$O_{\mathrm{ER}} = \sum_{\boldsymbol{x}\in\mathcal{D}} \underbrace{\sum_{\boldsymbol{a}\in\mathcal{B}^+(\boldsymbol{x})} R(\boldsymbol{a})\pi(\boldsymbol{a}\mid\boldsymbol{x})}_{\text{enumeration inside buffer}} + \underbrace{\sum_{\boldsymbol{a}\notin\mathcal{B}^+(\boldsymbol{x})} R(\boldsymbol{a})\pi(\boldsymbol{a}\mid\boldsymbol{x})}_{\text{expectation outside buffer}}. \tag{7}$$

Based on this observation, they propose to use enumeration to estimate the gradient of the first term on the RHS of (7) and use Monte Carlo sampling followed by rejection sampling to estimate the gradient of the second term on the RHS of (7) using the REINFORCE (Williams, 1992) estimator. This procedure is called Memory Augmented Policy Optimization (MAPO) and in its ideal form provides a low variance unbiased estimate of the gradient of (7) for deterministic $R(\cdot)$. Note that one can also incorporate entropy into MAPO (Liang et al., 2018) as the contribution of entropy can be absorbed into the reward function as $R'(\boldsymbol{a}) = R(\boldsymbol{a}) - \tau \log \pi(\boldsymbol{a}\mid\boldsymbol{x})$.

A.2 MODE COVERING EXPLORATION

Inspired by Norouzi et al. (2016); Nachum et al. (2017), we first note that the IML objective per context $\boldsymbol{x}$ can be expressed in terms of a KL divergence between an optimal policy π^* and the parametric policy π, *i.e.,* $\mathrm{KL}\left(\pi^* \parallel \pi\right)$, whereas the RER objective per context $\boldsymbol{x}$ can be expressed in terms of the same KL divergence, but *reversed*, *i.e.,* $\mathrm{KL}\left(\pi \parallel \pi^*\right)$. It is well understood that $\mathrm{KL}\left(\pi^* \parallel \pi\right)$ promotes *mode covering* behavior, whereas $\mathrm{KL}\left(\pi \parallel \pi^*\right)$ promotes mode seeking behavior. In other words, $\mathrm{KL}\left(\pi^* \parallel \pi\right)$ encourages all of the trajectories in $\mathcal{A}^+$ to have an equal probability, whereas RER, at least when $\tau = 0$, is only concerned with the *marginal* probability of successful trajectories and not with the way probability mass is distributed across $\mathcal{A}^+(\boldsymbol{x})$.

In Figure 3, we plot the fraction of contexts for which $|\mathcal{B}^+(\boldsymbol{x})| \geq k$, *i.e.,* the size of the buffer $\mathcal{B}^+(\boldsymbol{x})$ after convergence is larger than k as a function of k on two semantic parsing datasets and our results empirically validate the difference in the behaviour of these two objectives. For example, the fraction of context for which no plausible trajectory is found ($k = 10^0$ on the plots) is reduced by a few percent on both datasets, and for all other values of $k > 1$, the curve corresponding to IML is above the curve corresponding to MAPO, especially on WIKISQL.

When it comes to using O_{IML} (1) and O_{RER} (2) for learning from sparse feedback (*e.g.,* program synthesis) and comparing the empirical behavior of these different objective functions, there seems to be some disagreement among previous work. Abolafia et al. (2018) suggest that IML outperforms RER on their program synthesis problem, whereas Liang et al. (2017) assert that RER significantly outperforms IML on their weakly supervised semantic parsing problem. Examining the details of the experiments in Abolafia et al. (2018), we realize that their program synthesis tasks are primarily about discovering an individual program that is consistent with a few input-output examples. In this context, due to the presence of multiple input-output pairs, the issue of underspecified rewards poses a less serious challenge as compared to the issue of exploration. Hence, we believe that the success of IML in that context is consistent with our results in Figure 3.

A.3 LEARNING REWARDS WITHOUT DEMONSTRATION

Designing a reward function that distinguishes between optimal and suboptimal behavior is critical for the use of RL in real-world applications. This problem is particularly challenging when expert demonstrations are not available. When learning from underspecified success-failure rewards, one expects a considerable benefit from a refined reward function that differentiates different successful trajectories. While a policy $\pi(\boldsymbol{a}\mid\boldsymbol{x})$ optimized using a robust objective function such as RER and MML(Guu et al., 2017; Berant et al., 2013) learns its own internal preference between different successful trajectories, such a preference may be overly complex. This complexity arises particularly because the typical policies are autoregressive and only have limited access to trajectory level features.

Learning an auxiliary reward function presents an opportunity for using trajectory level features designed by experts to influence a preference among successful trajectories.

For instance, consider the problem of weakly-supervised semantic parsing, *i.e.,* learning a mapping from natural language questions to logical programs only based on the success-failure feedback for each question-program pair. In this problem, distinguishing between purposeful and accidental success without human supervision remains an open problem. We expect that one should be able to discount a fraction of the spurious programs by paying attention to trajectory-level features such as the length of the program and the relationships between the entities in the program and the question. The key technical question is how to combine different trajectory level features to build a useful auxiliary reward function.

Algorithm 1 Meta Reward-Learning (MeRL)

Input: $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, $\mathcal{B}^{+}_{\text{train}}$, $\mathcal{B}^{+}_{\text{val}}$
for step $t = 1, \ldots, \mathrm{T}$ **do**
  Sample batch of contexts $\mathcal{X}_{\text{train}}$ from $\mathcal{D}_{\text{train}}$ and $\mathcal{X}_{\text{val}}$ from $\mathcal{D}_{\text{val}}$
  Generate n_{explore} trajectories using $\pi_{\boldsymbol{\theta}}$ for each context in $\mathcal{X}_{\text{train}}$ and save successful trajectories to $\mathcal{B}^{+}_{\text{train}}$
  Compute $\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} O_{\text{train}}(\pi_{\boldsymbol{\theta}}, R_{\boldsymbol{\phi}})$ using samples from $(\mathcal{B}^{+}_{\text{train}}, \mathcal{X}_{\text{train}})$
  Compute $\boldsymbol{\phi}' = \boldsymbol{\phi} - \beta \nabla_{\boldsymbol{\phi}} O_{\text{val}}(\pi_{\boldsymbol{\theta}'})$ using samples from $(\mathcal{B}^{+}_{\text{val}}, \mathcal{X}_{\text{val}})$
  Update $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi}', \boldsymbol{\theta} \leftarrow \boldsymbol{\theta}'$
end for

Algorithm 2 Bayesian Optimization Reward-Learning (BoRL)

Input: $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$, $\mathcal{B}^{+}_{\text{train}}$
for trial $k = 1, \ldots, \mathrm{K}$ **do**
  Sample a parameter vector $\boldsymbol{\phi}_k$ for $R_{\boldsymbol{\phi}_k}$ by optimizing the acquisition function a_{M} over Bayesian model M i.e. $\boldsymbol{\phi}_k \leftarrow \underset{\boldsymbol{\phi}}{\operatorname{argmax}}\, a_M(\boldsymbol{\phi} \mid \mathcal{V}_{1:k-1})$
  Create a memory buffer $\mathcal{B}^{+}_{k}$ containing only the highest ranked trajectories in $\mathcal{B}^{+}_{\text{train}}$ based on $R_{\boldsymbol{\phi}_k}$
  for step $t = 1, \ldots, \mathrm{T}$ **do**
    Sample batch of contexts $\mathcal{X}_{\text{train}}$ from $\mathcal{D}_{\text{train}}$
    for context $\boldsymbol{c}$ in $\mathcal{X}_{\text{train}}$ **do**
      Generate n_{explore} trajectories $\mathcal{S}_{\boldsymbol{c}}$ using $\pi_{\boldsymbol{\theta}}$
      Save successful trajectories in $\mathcal{S}_{\boldsymbol{c}}$ ranked higher than any trajectory in $\mathcal{B}^{+}_{k}(\boldsymbol{c})$ based on $R_{\boldsymbol{\phi}_k}$
    end for
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} O_{\text{train}}(\pi_{\boldsymbol{\theta}})$ using samples from $(\mathcal{B}^{+}_{\text{train}}, \mathcal{X}_{\text{train}})$
  end for
  Evaluate v_k, the accuracy of policy π on $\mathcal{D}_{\text{val}}$
  Augment $\mathcal{V}_{1:k} = \{\mathcal{V}_{1:k-1}, (\boldsymbol{\phi}_k, v_k)\}$ and update the model M
end for

A.3.1 MERL *vs.* BORL

BoRL offers more flexibility than MeRL since we can optimize any non-differentiable objective on the validation set using BoRL but MeRL can only be used for differentiable objectives. Another advantage of BoRL over MeRL is that it performs global optimization over the reward parameters as compared to the local gradient based optimization in MeRL. Notably, the modular nature of Bayesian optimization and the widespread availability of open source libraries for black box optimization makes BoRL much more easier to implement than MeRL. However, MeRL is much more computationally efficient that BoRL due to having access to the gradients of the objective to optimize. Additionally, MeRL has the ability to adapt the auxiliary rewards throughout the course of policy optimization while BoRL can only express reward functions that remain fixed during policy optimization.

A.4 Related Work

The problem we study in this work as well as the proposed approach intersect with many subfields of machine learning and natural language processing discussed separately below.

Reward learning. Reinforcement learning (RL) problems are specified in terms of a reward function over state-action pairs, or a trajectory return function for problems with sparse feedback. A key challenge in applying RL algorithms to real world problems is the limited availability of a rich and reliable reward function. Prior work has proposed to learn the reward function (1) from expert demonstrations using inverse reinforcement learning (Abbeel & Ng, 2004; Ziebart et al., 2008) or adversarial imitation learning (Ho & Ermon, 2016) and (2) from human feedback (Christiano et al., 2017; Leike et al., 2018; Ibarz et al., 2018). Recently, these ideas have been applied to the automatic discovery of goal specifications (Xie et al., 2018; Bahdanau et al., 2019), text generation tasks (Wang et al., 2018; Wu et al., 2017; Bosselut et al., 2018) and the optimization of reward functions (*e.g.,* Gleave & Habryka (2018); Fu et al. (2019); Shi et al. (2018)) via inverse RL. By contrast, we aim to learn a reward function through meta-learning to enhance underspecified rewards without using any form of trajectory or goal demonstrations. Another relevant work is LIRPG (Zheng et al., 2018), which learns a parametric intrinsic reward function that can be added to the extrinsic reward to improve the performance of policy gradient methods. While the intrinsic reward function in LIRPG is trained to optimize the extrinsic reward, our reward function is trained to optimize the validation set performance through meta-learning, because our main concern is generalization.

Meta-learning. Meta-learning aims to design learning algorithms that can quickly adapt to new tasks or acquire new skills, which has shown recent success in RL (Finn et al., 2017; Duan et al., 2016; Wang et al., 2016; Nichol & Schulman, 2018). There has been a recent surge of interest in the field of meta-reinforcement learning with previous work tackling problems such as automatically acquiring intrinsic motivation (Zheng et al., 2018), discovering exploration strategies (Gupta et al., 2018; Xu et al., 2018b), and adapting the nature of returns in RL (Xu et al., 2018c). It has also been applied to few-shot inverse reinforcement learning (Xu et al., 2018a), online learning for continual adaptation (Nagabandi et al., 2018), and semantic parsing by treating each query as a separate task (Huang et al., 2018). Concurrent work (Zou et al., 2019) also dealt with the problem of learning shaped rewards via meta-learning. Recent work has also applied meta-learning to reweight learning examples (Ren et al., 2018) to enable robust supervised learning with noisy labels, learning dynamic loss functions (Wu et al., 2018) and predicting auxiliary labels (Liu et al., 2019) for improving generalization performance in supervised learning. In a similar spirit, we use meta optimization to learn a reward function by maximizing the generalization accuracy of the agent's policy. Our hypothesis is that the learned reward function will weight correct trajectories more than the spurious ones leading to improved generalization.

Semantic parsing. Semantic parsing has been a long-standing goal for language understanding (Winograd, 1972; Zelle & Mooney, 1996; Chen & Mooney, 2011). Recently, weakly supervised semantic parsing (Berant et al., 2013; Artzi & Zettlemoyer, 2013) has been proposed to alleviate the burden of providing gold programs or logical forms as annotations. However, learning from weak supervision raises two main challenges (Berant et al., 2013; Pasupat & Liang, 2016a; Guu et al., 2017): (1) how to explore an exponentially large search space to find gold programs; (2) how to learn robustly given spurious programs that accidentally obtain the right answer for the wrong reason. Previous work (Pasupat & Liang, 2016b; Mudrakarta et al., 2018; Krishnamurthy et al., 2017) has shown that efficient exploration of the search space and pruning the spurious programs by collecting more human annotations has a significant impact on final performance. Some recent work (Berant et al., 2019; Cho et al., 2018) augments weak supervision with other forms supervisions, such as user feedback or intermediate results. Recent RL approaches (Liang et al., 2017; 2018) rely on maximizing expected reward with a memory buffer and performing systematic search space exploration to address the two challenges. This paper takes such an approach a step further, by learning a reward function that can differentiate between spurious and correct programs, in addition to improving the exploration behavior.

Language grounding. Language grounding is another important testbed for language understanding. Recent efforts includes visual question answering (Antol et al., 2015) and instruction following in simulated environments (Hermann et al., 2017; Chevalier-Boisvert et al., 2019). These tasks usually focus on the integration of visual and language components, but the language inputs are usually automatically generated or simplified. In our experiments, we go beyond simplified environments,

and also demonstrate significant improvements in real world semantic parsing benchmarks that involve complex language inputs.

A.5 SEMANTIC PARSING

A.5.1 DATASETS

WIKITABLEQUESTIONS (Pasupat & Liang, 2015) contains tables extracted from Wikipedia and question-answer pairs about the tables. There are 2,108 tables and 18,496 question-answer pairs splitted into train/dev/test set. We follow the construction in (Pasupat & Liang, 2015) for converting a table into a directed graph that can be queried, where rows and cells are converted to graph nodes while column names become labeled directed edges. For the questions, we use string match to identify phrases that appear in the table. We also identify numbers and dates using the CoreNLP annotation released with the dataset.

The task is challenging in several aspects. First, the tables are taken from Wikipedia and cover a wide range of topics. Second, at test time, new tables that contain unseen column names appear. Third, the table contents are not normalized as in knowledge-bases like Freebase, so there are noises and ambiguities in the table annotation. Last, the semantics are more complex comparing to previous datasets like WEBQUESTIONSSP (Yih et al., 2016). It requires multiple-step reasoning using a large set of functions, including comparisons, superlatives, aggregations, and arithmetic operations (Pasupat & Liang, 2015).

Figure 4 shows some natural language queries in WIKITABLEQUESTIONS for which both the models trained using MAPO and MeRL generated the correct answers despite generating different programs.

WIKISQL (Zhong et al., 2017) is a recent large scale dataset on learning natural language interfaces for databases. It also uses tables extracted from Wikipedia, but is much larger and is annotated with programs (SQL). There are 24,241 tables and 80,654 question-program pairs splitted into train/dev/test set. Comparing to WIKITABLEQUESTIONS, the semantics are simpler because SQL use fewer operators (column selection, aggregation, and conditions). We perform similar preprocessing as for WIKITABLEQUESTIONS. We don't use the annotated programs in our experiments.

Example	Comment
Query nu-1167: **Who was the first oldest living president?** MAPO: v_0 = (first all_rows); v_{ans} = (hop v_0 r.president) MeRL: v_0 = (argmin all_rows r.became_oldest_living_president-date); v_{ans} = (hop v_0 r.president)	Both programs generate the correct answer despite MAPO's program being spurious since it assumes the database table to be sorted based on the *became_oldest_living_president-date* column.
Query nu-346: **What tree is the most dense in India?** MAPO: v_0 = (argmax all_rows r.density); v_{ans} = (hop v_0 r.common_name) MeRL: v_0 = (filter_str_contain_any all_rows [u'india'] r.location); v_1 = (argmax v_0 r.density); v_{ans} = (hop v_1 r.common_name)	MAPO's program generates the correct answer by chance since it finds the tree with most density which also happens to be in India in this specific example.
Query nu-2113: **How many languages has at least 20,000 speakers as of the year 2001?** MeRL: v_0 = (filter_ge all_rows [20000] r.2001_..._number); v_{ans} = (count v_0) MAPO: v_0 = (filter_greater all_rows [20000] r.2001_..._number); v_{ans} = (count v_0)	Since the query uses "at least", MeRL uses the correct function token *filter_ge* (i.e $\geq$ operator) while MAPO uses *filter_greater* (i.e. $>$ operator) which accidentally gives the right answer in this case. For brevity, *r.2001_..._number* refers to *r.2001_census_1_total_population_1_004_59_million-number*.

Figure 4: Example of generated programs from models trained using MAPO and MeRL on WIKITABLEQUESTIONS

A.5.2 AUXILIARY REWARD FEATURES

In our semantic parsing experiments, we used the same preprocessing as implemented in MAPO. The natural language queries are preprocessed to identify numbers and date-time entities. In addition,

Table 5: MAPOX hypeparameters used for experiments in Table 2.

Hyperparameter	Value
Entropy Regularization	9.86 x 10^{-2}
Learning Rate	4 x 10^{-4}
Dropout	2.5 x 10^{-1}

Table 6: BoRL hypeparameters used in experiments in Table 2.

Hyperparameter	Value
Entropy Regularization	5 x 10^{-2}
Learning Rate	5 x 10^{-3}
Dropout	3 x 10^{-1}

Table 7: MeRL hypeparameters used in experiments in Table 2.

Hyperparameter	Value
Entropy Regularization	4.63 x 10^{-2}
Learning Rate	2.58 x 10^{-2}
Dropout	2.5 x 10^{-1}
Meta-Learning Rate	2.5 x 10^{-3}

phrases in the query that appear in the table entries are converted to string entities and the columns in the table that have a phrase match are assigned a column feature weight based on the match.

We used the following features for our auxiliary reward for both WIKITABLEQUESTIONS and WIKISQL:

- $\boldsymbol{f}_1$: Fraction of total entities in the program weighted by the entity length
- $\boldsymbol{f}_2$, $\boldsymbol{f}_3$, $\boldsymbol{f}_4$: Fraction of date-time, string and number entities in the program weighted by the entity length respectively
- $\boldsymbol{f}_5$: Fraction of total entities in the program
- $\boldsymbol{f}_6$: Fraction of longest entities in the program
- $\boldsymbol{f}_7$: Fraction of columns in the program weighted by the column weight
- $\boldsymbol{f}_8$: Fraction of total columns in the program with non-zero column weight
- $\boldsymbol{f}_9$: Fraction of columns used in the program with the highest column column weight
- $\boldsymbol{f}_{10}$: Fractional number of expressions in the program
- $\boldsymbol{f}_{11}$: Sum of entities and columns weighted by their length and column weight respectively divided by the number of expressions in the program

A.5.3 TRAINING DETAILS

Our implementation is based on the open source implementation of MAPO (Liang et al., 2018) in Tensorflow (Abadi et al., 2016). We use the same model architecture as MAPO which combines a seq2seq model augmented by a key-variable memory (Liang et al., 2017) with a domain specific language interpreter. We utilized the hyperparameter tuning service (Golovin et al., 2017) provided by Google Cloud for BoRL.

We used the optimal hyperparameter settings for training the vanilla IML and MAPO provided in the open source implementation of MAPO. One major difference was that we used a single actor for our policy gradient implementation as opposed to the distributed sampling implemented in Memory Augmented Program Synthesis.

For our WIKITABLEQUESTIONS experiments in Table 2, we initialized our policy from a pretrained MAPO checkpoint (except for vanilla IML and MAPO) while for all our WIKISQL experiments, we trained the agent's policy starting from random initialization.

For the methods which optimize the validation accuracy using the auxiliary reward, we trained the auxiliary reward parameters for a fixed policy initialization and then evaluated the top K hyperparam-

Table 8: MAPOX hypeparameters used for experiments in Table 3.

Hyperparameter	Value
Entropy Regularization	5.1 x 10^{-3}
Learning Rate	1.1 x 10^{-3}

Table 9: BoRL hypeparameters used in experiments in Table 3.

Hyperparameter	Value
Entropy Regularization	2 x 10^{-3}
Learning Rate	1 x 10^{-3}

Table 10: MeRL hypeparameters used in experiments in Table 3.

Hyperparameter	Value
Entropy Regularization	6.9 x 10^{-3}
Learning Rate	1.5 x 10^{-3}
Meta-Learning Rate	6.4 x 10^{-4}

eter settings 5 times (starting from random initialization for WIKISQL or on 5 different pretrained MAPO checkpoints for WIKITABLEQUESTIONS) and picked the hyperparameter setting with the best average validation accuracy on the 5 runs to avoid the danger of overfitting on the validation set.

We only used a single run of IML for both WIKISQL and WIKITABLEQUESTIONS for collecting the exploration trajectories. For WikiSQL, we used greedy exploration with one exploration sample per context during training. We run the best hyperparameter setting for 10k epochs for both WIKISQL and WIKITABLEQUESTIONS. Similar to MAPO, the ensembling results reported in Table 4 used 5 different training/validation splits of the WIKITABLEQUESTIONS dataset. This required training 5 different IML models on each split to collect the exploration trajectories.

We ran BoRL for 384 trials for WIKISQL and 512 trials for WIKITABLEQUESTIONS respectively. We used random search with 30 different settings to obtain the optimal hyperparameter values for all our experiments. The detailed hyperparameter settings for WIKITABLEQUESTIONS and WIKISQL experiments are listed in Table 5 to Table 7 and Table 8 to Table 10 respectively. Note that we used a dropout value of 0.1 for all our experiments on WIKISQL except MAPO which used the optimal hyperparameters reported by Liang et al. (2018).

A.6 INSTRUCTION FOLLOWING TASK

This task is composed of a simple instruction following environment in the form of a simple maze of size $N \times N$ with K deadly traps distributed randomly over the maze. A goal located in one of the four corners of the maze (see Figure 1a). An agent is provided with a language instruction, which outlines an optimal path that if the agent takes it can reach the goal without being trapped. The agent receives a reward of 1 if it succeeds in reaching the goal within a certain number of steps, otherwise 0.

To increase the difficulty of this task, we reverse the instruction sequence that the agent receives, *i.e.,* the command "Left Up Right" corresponds to the optimal trajectory of actions $(\rightarrow, \uparrow, \leftarrow)$. We use a set of 300 randomly generated environments with $(N, K) = (7, 14)$ with training and validation splits of 80% and 20% respectively. The agent is evaluated on 300 unseen test environments from the same distribution. To mitigate the issues due to exploration, we train the agent using a fixed replay buffer containing the gold trajectory for each environment.

A.6.1 AUXILIARY REWARD FEATURES

In the instruction following task, the auxiliary reward function was computed using the single and pairwise comparison of counts of symbols and actions in the language command $\boldsymbol{x}$ and agent's trajectory $\boldsymbol{a}$ respectively. Specifically, we created a feature vector $\boldsymbol{f}$ of size 272 containing binary features of the form $f(a, c) = \#_a(\boldsymbol{x}) == \#_c(\boldsymbol{a})$ and $\boldsymbol{f}(ab, cd) = \#_{ab}(\boldsymbol{x}) == \#_{cd}(\boldsymbol{a})$ where

Table 11: MAPO hypeparameters used for the setup with Oracle rewards in Table 1.

Hyperparameter	Value
Entropy Regularization	3.39 x 10^{-2}
Learning Rate	5.4 x 10^{-3}

Table 12: MAPO hypeparameters used for the setup with underspecifed rewards in Table 1.

Hyperparameter	Value
Entropy Regularization	1.32 x 10^{-2}
Learning Rate	9.3 x 10^{-3}

Table 13: MeRL hypeparameters used for the setup with underspecifed + auxiliary rewards in Table 1.

Hyperparameter	Value
Entropy Regularization	2 x 10^{-4}
Learning Rate	4.2 x 10^{-2}
Meta-Learning Rate	1.5 x 10^{-4}
Gradient Clipping	1 x 10^{-2}

$a, b \in \{$Left, Right, Up, Down$\}$ and $c, d \in \{0, 1, 2, 3\}$ and $\#_i(\boldsymbol{j})$ represents the count of element i in the vector $\boldsymbol{j}$. We learn one weight parameter for each single count comparison feature. The weights for the pairwise features are represented using the weights for single comparison features $\boldsymbol{w}_{(ab,cd)} = \alpha * \boldsymbol{w}_{ac} * \boldsymbol{w}_{bd} + \beta * \boldsymbol{w}_{ad} * \boldsymbol{w}_{bc}$ using the additional weights α and β.

The auxiliary reward is a linear function of the weight parameters. However, in case of MeRL, we also used a softmax transformation of the linear auxiliary reward computed over all the possible trajectories (atmost 10) for a given language instruction.

A.6.2 Training Details

We used the Adam Optimizer (Kingma & Ba, 2014) for all the setups with a replay buffer memory weight clipping of 0.1 and full-batch training. We performed hyperparameter sweeps via random search over the interval (10^{-4}, 10^{-2}) for learning rate and meta-learning rate and the interval (10^{-4}, 10^{-1}) for entropy regularization. For our MeRL setup with auxiliary + underspecified rewards, we initialize the policy network using the MAPO baseline trained with the underespecified rewards. The hyperparameter settings are listed in Table 11 to Table 13. MeRL was trained for 5000 epochs while other setups were trained for 8000 epochs. We used 2064 trials for our BoRL setup which was approximately 20x the number of trials we used to tune hyperparameters for other setups.