We thank all reviewers for valuable comments. We commit to improving clarity of definitions/approximations/algorithm details and add more discussions on related works in the camera-ready version.

**Usage of entropy:** Entropy is used to measure sufficiency, compactness and uniqueness. Sufficiency is measured by \( H(r_i|s_i,a) \) in def.1, where the sufficient sub-state set \( M_i \) represents all sub-states \( \hat{s}_i \) that are as informative as the whole state \( s \) in terms of inferring \( r_i \). Compactness is measured by \( H(s_i) \) in def.1&2, where \( C \) represents all sets of sub-rewards (and corresponding sub-states) that is non-trivial. Uniqueness/diversity) is measured by \( H(s_i|s_j) \), and \( H(r_i|s_j,a) \) as an alternative. One may argue that, it is easier to use feature number to capture compactness and uniqueness, for example using \( |s_i - s_j| \) to capture diversity/uniqueness. This is a good and simple formulation under factored MDP in Section 3 when all features are independent. However, for features learnt by networks, independence is not guaranteed and even when \( m_i \) and \( m_j \) does not overlap, the mutual information between \( s_i \) and \( s_j \) could still be high. The usage of entropy \( (H(s_i|s_j) \) and \( H(r_i|s_j,a) \)) allows us to discourage such case while \( |s_i - s_i \cap s_j| \) cannot.

**Explanation of \( L_{div1} \):** \( L_{div1} \) computes the sum of \( H(\hat{s}_i|\hat{s}_j) \), which can be interpreted as randomness of sub-state \( \hat{s}_i \) given sub-state \( \hat{s}_j \). To further explain the intuition behind, consider a factored MDP where a factor is either chosen or not chosen for each sub-states. Note that a factor \( x_k \) will only contribute to \( H(\hat{s}_i|\hat{s}_j) \) if \( x_k \) is chosen by \( \hat{s}_i \) and not chosen by \( \hat{s}_j \), i.e. \( m_{i,k} = 1 \) and \( m_{j,k} = 0 \). A simple way to extend this boolean expression is to use ReLU \( (m_{i,k} - m_{j,k}) \). We admit that the approximation \( L_{div1} \) for \( H(s_i|s_j) \) does not deal with the correlated case of \( s_i \) and \( s_j \) as well as \( L_{div2} \), which may explain the good performance of \( L_{div2} \) over \( L_{div1} \) in Atari Games where the feature could be correlated rather than independent as in well-defined factored MDP (e.g. our toy case).

**Explanation of \( L_{div2} \):** The usage of variance to approximate entropy was discussed in L203. Note the definition of variance \( \text{Var}(r_i|s_j,a) = \mathbb{E}[(r_i - \mathbb{E}[r_i|s_j,a])^2] \). To obtain an estimation for \( \mathbb{E}[r_i|s_j,a] \), we use a network \( \hat{r}_i = g_{\theta}(\hat{s}_j,a) \) and minimize \( \text{MSE}(r_i,\hat{r}_i) \) over parameter \( \theta_i \). Then we can use \( \hat{r}_i \) as an estimation for \( \mathbb{E}[r_i|s_j,a] \) and \( \text{MSE}(r_i,\hat{r}_i) \) as a surrogate for \( \text{Var}(r_i|s_j,a) \) and maximize \( \text{MSE}(r_i,\hat{r}_i) \) over \( \hat{s}_j \) to increase variance/entropy. We apologize for the ambiguity and will refine it in the camera-ready version.

**Downstream sub-Q learning:** The detailed version of RD^2 algorithm can be found in Appendix A. In brief, sub-Q functions are trained with both full reward TD and sub-reward TD. The usage of global action \( a_{t+1} \) instead of local actions (i.e. \( a_{t+1} = \arg\max_{a} Q(s_{t+1},a) \) assures invariant optimal Q-function \( Q^* \).

**Ablation study for each loss term:** To investigate the contribution of each loss term, we show that ablative performance. Specifically, we compare three variants of RD^2: (1) RD^2 without \( L_{sum} \) in Eq.4; (2) RD^2 without \( L_{mini} \) in Eq.5; (3) RD^2 without \( L_{div2} \) in Eq.7. As shown in Figure[1] when we drop the \( L_{sum} \) term, RD^2 is equivalent to learn with randomly decomposed reward. Therefore, the performance deteriorates dramatically. When we drop the diversity encouraging term \( L_{div2} \), we get the half-half reward decomposition, which is not helpful to accelerate the training process. Finally, we find that the minimal sufficient regularization term \( L_{mini} \) mainly contributes to the later training process.

**To Reviewer 1:** Q1: Dynamics blind. A1: Decomposing dynamics is also an interesting topic that we would love to look into, however it may require stricter assumptions on the environment. Q2: How were the games for Atari chosen? A2: We follow prior work [Lin et al.’19] and test our algorithm on the Atari games that have multiple sources of reward. We will run our algorithm in more environments and provide the results in Appendix.

**To Reviewer 2:** Q1: Beyond K=2. A1: We found that in environments with more than two reward sources, using K>2 will achieve better performance. Moving beyond prior info about K, self-tuning K would be an interesting future work.

**To Reviewer 3:** Q1: About the runtime of estimation of approximating loss. A1: Despite the estimation of approximating loss, our efficient implementation can train at roughly 80% of Rainbow’s speed. Q2: Sensitivity to hyperparameters. A2: We provide the hyperparameter search range in appendix B. In practice, we found that our algorithm can work well if the value of hyperparameters are in a reasonable range. For example, on one hand, since the sub-Q loss and \( L_{mini} \) serve as regularization terms, we set their corresponding learning rate to a relatively small value; on the other hand, we keep the learning rate of \( L_{sum} \) and \( L_{div2} \) in the same scale of original Rainbow. Overall, our algorithm is not sensitive to the hyperparameters.

**To Reviewer 4:** Q1: The use of the property \( H(cX) = H(X) + \log(|c|) \). A1: We are aware that this does not apply when \( c \) is dependent on \( X \). The cause of this gap is that we let \( m_i \) (i.e. chosen factors) be dependent on \( s \), while in section 3 \( s_i \) is fixed. If we dig deeper, the root of this gap is that features can not be viewed as factors. A factor could be \( x \) coordinate of the agent, but without additional supervision it is impossible for networks to extract such compact information. One way to view features is to see them as index-varying factors. E.g., at timestep \( t \) a feature could be \( \{x_1,x_2,x_3\} \) but at timestep \( t+1 \) it could be \( \{x_3,x_1,x_2\} \). Then we can let \( m_i \) be fixed and introduce a permutation matrix \( P(s) \) that is dependent on \( s \) and let sub-state \( s_i = m_iP(s) \cap f(s) \). It is easy to show that \( H(m_iP(X) \cap X) = H(X) + \log(|c|) \).

However, we did not implement the permutation form in our paper, mainly due to that there are still flaws in the index-varying factor perspective of features and that current RD^2 has already achieved significant performance.