Thank you for your excellent feedback. Solving Partially Observable RL is challenging and, as noted by R3, the proposed method had much better performance than A3C, ACER, and PPO with LSTM. We address your main questions here and will elaborate in the paper.

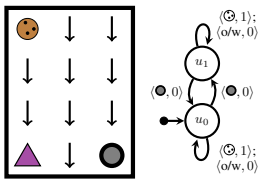

Figure 1: Gravity domain

How well does Tabu search scale? (R1): Roughly speaking, Tabu search scales as well as genetic algorithms (which are commonly used in RL). Its main bottleneck is evaluating the neighbourhood around the current RM solution, but this step is easily parallelizable. Given a training set composed of 1 million transitions, a simple Python implementation of Tabu search took less than 2.5 minutes to learn an RM across all our environments (using 62 workers). In our experiments, the agent relearned the RM 8.6 times (on average) per run. Note that the size of the neighbourhood depends on the number of possible abstract observations, and so exhaustively evaluating the neighbourhood may sometimes become impractical. This well-studied problem has plenty of proposed solutions (known as Large Neighborhood Search methods), though.

What are the limitations of LRM (R1) and when might QRM not work as intended (R2)?: As R1 mentioned, an interesting idea from LRM was to optimize over a necessary condition for perfect RMs. This objective favors RMs that are able to predict possible and impossible future observations at the abstract level given by the labelling function L. Learning the RM at the abstract level is efficient but requires ignoring (possibly relevant) low-level information. (In the future, we would like to learn LSTM policies inside the RM to account for any missing information in L.)

To the limitations, Figure 1 shows an adversarial example for LRM. The agent receives reward for eating the cookie (🍪). There is an external force pulling the agent down—i.e., the outcome of the "`move up`" action is actually a downward movement with high probability. There is a button (🔘) that the agent can press to turn off (or back on) the external force. Hence, the optimal policy is to press the button and then eat the cookie. Given $\mathcal{P} = \{🍪, 🔘\}$, a perfect RM for this environment is fairly simple (see Figure 1) but LRM might not find it. The reason is that pressing the button changes the low-level probabilities in the environment but does not change what is possible or impossible at the abstract level. Moreover, if a perfect RM is found, our heuristic approach to share experiences in QRM would not work as intended because the experiences collected when the force is on (at u_0) would be used to update the policy for the case where the force is off (at u_1). From a practical perspective, a simple solution is to add a high-level detector that senses the external force. Nonetheless, we will discuss the theoretical implications of this interesting adversarial example in the paper.

Question about the evaluation metric and grid sizes (R1): Indeed, we are reporting the maximum "*cumulative*" rewards for the baselines every $10,000$ steps. The idea was to highlight that no run of the baselines learned to consistently solve the tasks. We used small grids in our experiments (it takes 8 steps to go from one room to another). Hence, a 10-order memory for DDQN should be enough (in principle) to learn to solve our environments.

Relation with Zhang et al.'s work (R1): This interesting work—which was submitted to arxiv after the NeurIPS deadline—will definitely be discussed in our paper. In short, the main distinction is that LRM exploits a high-level abstraction of the observations to understand how to decompose the problem into Markovian subproblems. Exploiting abstractions has historically helped RL agents to solve challenging domains. In contrast, Zhang et al.'s work has the merit of learning PSRs at the low-level, making it easier to use as an off-the-shelf tool for Partially Observable RL.

What if the high-level detectors were noisy? (R2): To handle noise over L (without requiring an explosion of the size of the RM), it seems necessary to move from deterministic to stochastic RMs. This is to allow the agent to be at multiple RM states with certain probability. We believe this is an interesting research direction.

Are you planning to release your code? Does LRM+DDQN/DQRN require a stochastic environment? Could you add more empirical evaluation on why LRM+DQRM converges faster than LRM+DDQN? (R2): We will release our code. Our approach works well on the deterministic version of our environments too. We will add further information, including exploration heatmaps and learned trajectories, to the supplementary materials.

How to prepare an effective labelling function L? (R3): Intuitively, any event that might be useful for the agent to remember is a good candidate to be included in L. We are also interested in investigating methods for learning a suitable L during environment interaction, and feel this paper demonstrates how one can be exploited if learned (or given).

Is LRM a model-based RL approach and, as such, shouldn't it be compared with Doshi-Velez et al.? (R3): Thanks for pointing this out. LRM+DDQN/DQRM lies somewhere between model-based and model-free as the RM is learned in a model-based fashion but its policies are learned in a model-free way. This allows our models to leverage deep RL's ability to learn policies from low-level inputs (e.g., images). As such, our baselines and part of the related work discussion was indeed biased towards deep RL approaches for Partially Observable RL. We will partially remedy this by including a discussion about non-Parametric methods, including Doshi-Velez et al., in the related work section.

Clarification in proof sketch of Theorem 4.3 (R3): The discrepancy between Def 4.1 and the LRM's objective value comes from the fact that LRM is optimizing over a necessary (but not sufficient) condition for finding a perfect RM. If this does not answer your question, please let us know and we will further elaborate on a revised version of this work.