We thank the reviewers for their helpful suggestions, and will do our best to incorporate them into our paper. Overall, we want to emphasize that the goal of SPECTRL is to make it easier to apply RL to tasks with complex objectives. In particular, SPECTRL enables the user *program* what the agent needs to do at a high level; then, SPECTRL automatically learns a policy that tries to best satisfy the user intent.

R1, LSTMs: It is true that LSTMs can be used to solve RL problems with non-Markovian specifications. However, the task monitor introduced in this paper not only eliminates the need to learn the internal state, but also enables us to perform reward shaping. As we show in our experiments, reward shaping is crucial for learning complex tasks. Thus, even if the LSTM learns a perfect encoding of the state, our approach would still substantially outperform it.

R2, Curiosity-driven exploration: These approaches aren't really applicable to addressing our challenges, which involve complex objectives rather than hard exploration. For example, the task may be for the agent to visit a sequence of goals in a particular order. The agent may know how to reach each of these goals individually, but this is not sufficient for the agent to complete the task.

R2, Sub-goal based rewards. Many challenges arise when considering the details of how sub-goal based rewards would be implemented. For example, how does achieving a sub-goal count compared to violating a constraint? How do we handle sub-goals that can be achieved in multiple ways? How do we ensure the agent does not repeatedly obtain a reward for a sub-goal that it has already completed? As tasks get more complex/deeply nested, manually determining rewards for sub-goals is very non-trivial. These challenges are exactly what our system is designed to solve.

R2, Baselines: We used ARS since it has been demonstrated to be state-of-the-art for continuous control. For fair comparison, we use ARS with TLTL as well. Also, like TLTL, our approach can be used with with other algorithms.

R2, More challenging environments: Our experiments already show that our algorithm can solve problems that the state-of-the-art ARS algorithm cannot. Benchmarks such as MuJoCo may involve more complex dynamics, but they focus largely on short-term control tasks—e.g., running in a straight line as fast as possible. In contrast, the benefit of our approach is on handling complex, long-term control tasks, where the agent must perform a variety of actions to achieve complex objectives—e.g., reaching multiple goals in sequence and then coming back. While there has been some success solving these problems using traditional RL, these approaches typically rely on enormous amounts of computation. In contrast, our algorithm converges quickly without a large amount of computation (all our experiments are run on a single GPU).

R2, Cart-Pole: Our cart-pole benchmark differs from the standard OpenAI one in two ways. (i) The goal in the standard one is just to keep the pole balanced for as long as possible. In ours, the goal is to move the cart to the right and then back to the starting position without letting the pole fall. We find this task to be substantially more challenging than the standard one. (ii) The standard benchmark has discrete actions, whereas ours has continuous actions, which also makes the task more challenging.

R2, TLTL: We have two contributions over TLTL. (i) While it is possible to extend the state space to handle non-Markovian TLTL specifications, this could not previously be done automatically. One of our contributions is to provide a way for automatically extending the state space; our algorithm can be modified to work for TLTL formulas as well. (ii) We perform reward shaping, whereas TLTL does not. We find that reward shaping is critical for good performance.

R3, Reward machines: We differ from the paper on reward machines in two ways. First, in their paper, the specification is given directly as a finite state machine along with reward functions for each state. Their key contribution is in the ability to learn multiple tasks simultaneously by applying the Q-learning updates across different specifications. In contrast, our contribution is to automatically generate a task monitor from a high-level logical specification. Second, our notion of task monitor is more expressive since it has a finite set of registers that can store real values. In contrast, finite state machines cannot store quantitative information. We will add a discussion to our related work.

R3, barriers: Some of our benchmarks include barriers (i.e., obstacles). Our approach is able to solve these tasks.

Presentation: To improve readability, we plan to add pictures describing the high level approach in a final version. We will include error-bar plots for Figure 3, which we show below. To exclude outliers, we omitted one best and one worst run out of the 5 runs. While the variance is sometimes high, our insights continue to hold—in particular, our algorithm consistently outperforms the baseliness. We will add a conclusion as well.