First, we would like to thank the reviewers for their helpful feedback. Here, we address the main questions, and we will clarify all of these points in any future revision.

Reviewers 1, 2, 3, and 4:

- **It is difficult/impossible to generate a worst case environment model in many scenarios.** A formal proof of an agent's safety is a worst-case guarantee on states that will be reached at future time points. Such states are not well-defined without a worst-case environment model. For this reason, all prior work on formal approaches to safety in reinforcement learning (Alshiekh et al., 2018; Fulton & Platzer, 2018; Zhu et al., 2019) have required such models. As for how easy it is to write such models, note that the model only needs to capture worst-case constraints on the MDP, as opposed to the entirety of the agent's dynamics. For example, our adaptive cruise control benchmark uses a model of cars on a road and only constrains the maximum braking and accelerating force which can be generated by the cars. Writing such worst-case constraints by hand is often feasible. It may be possible to learn likely worst-case models through exploration, but this is a challenging orthogonal problem, and we leave it for future work.
- **How efficient is our algorithm?** The training time for our technique is typically about 8-10 hours using a 6-core desktop machine (and a single thread for shield updates). To clarify, our claim about efficiency is compared to approaches that provide guaranteed safety during training. Verifying purely neural representations in this context is prohibitively expensive. In order to validate this claim, we attempted to verify a network in one of our simpler benchmarks using Marabou (Katz et al., 2019), a state-of-the-art neural network analysis tool. We were unable to verify a property within half an hour. Since we need to make thousands of calls to the verification oracle during training, this indicates that a purely neural approach is infeasible for safe reinforcement learning. We will be happy to add more details on this in the paper's final version.

Reviewer 2:

- **Formal methods aspects of the paper are terse and hard to understand for the intended audience**. We will expand the relevant parts of the paper to give better intuition for the formal methods components of our algorithm.
- **Can shield synthesis scale to more complex state spaces?** Scalability depends on the number of regions in the piecewise linear policies (which the user can control) and underlying tools for formally verifying piecewise linear functions. These tools are advancing rapidly, and our framework can make use of new developments in this area.

Reviewer 3:

- **How is safety maintained by using imitation learning on the neural component of the policy?** Our policies have the form "if ϕ then f else g" (f is the neural component, g is the shield), and the safety of the policy only depends on ϕ and g. The imitation learning part affects f, but not g or ϕ, and doesn't compromise the property above. Thus, the neurosymbolic policy is still safe, and therefore there are no safety violations in the experiments.
- **How often is the shield invoked?** Typically once training has finished the shield is not invoked. For example the "pendulum", "acc", and "mid-obstacle" benchmarks all have zero shield interventions after training. Intuitively this is because the network learns that running into the shield yields a lower reward. Earlier in training while the network is taking riskier exploration steps the shield is invoked more often, sometimes in as much as 20% of the time.
- **Why does the static shielding approach seem to outperform Revel on the mountain-car benchmark?** There appears to be a misunderstanding. In our plots, we show a loss rather than a reward. Thus lower is better on our plots, and Revel is significantly better than the static shielding approach.

Reviewer 4:

- **How do we initialize the shield and monitor?** In our experiments, the initial shields are trivial shields that one can easily write by hand. For example, in the obstacle benchmark, the shield never moves the controlled robot. In principle, we could adapt approaches from prior work (Fulton & Platzer, 2018; Zhu et al. 2019) to generate an initial shield automatically. However, this is itself a challenging, orthogonal question that is best left to future work.
- **How do we compute the monitor?** Our shield update algorithm produces a shield which is safe for at least the same states as the original shield. We then iteratively expand the monitor until we can no longer verify safety.
- **Theoretical complexity of the shield synthesis algorithm is not presented.** The shield synthesis algorithm used in the experiments is a form of imitation learning and runs for a chosen number of iterations before being cut off.
- **How is this approach different from minimizing the expected reward where unsafe states are given a large cost?** We prove that the system will *never* take any unsafe action (as opposed to giving a guarantee in expectation). In contrast, prior techniques that use the reward function to encode safety constraints must always explore a state at least once to measure the reward of that state. Specifically, MuZero relies on a learned prediction function which would itself need to be verified against the model in order to get the kind of guarantees we provide.
- **You only evaluate on piecewise linear policies but claim that the approach handles deep neural policies.** The policies in our experiments have the form "if ϕ then f else g" where g is piecewise-linear and f is a neural network. We could replace g with a different class of policies, but g could not be a neural network because existing approaches for neural network for verification are not efficient enough to be incorporated into our safe exploration method.