(R1) Extension to POMDP domains. We appreciate the insightful question. We are indeed planning to generalize RPN to POMDP domains. The POMDP setting is challenging for both classical planning and modern learning-based planning mainly because it requires *active sensing* steps [19]. Consider the example goal `InHand(Apple)` when the apple is in a closed fridge. To learn to plan for such a goal within the RPN framework, we may assume that the video demonstration includes sensing intermediate goals, e.g., a human demonstrator searches for the apple at a few locations including the fridge. Given such demonstration data, we can train an RPN to predict these sensing goals (e.g., `IsOpen(Fridge)`) as the *preconditions* of achieving the final goal. However, more principled approaches [19] may require extending RPN to explicitly reason about uncertainty of the state, which is out of the scope of this paper. We will revise the current running example to avoid possible confusion as suggested by the reviewer.

(R1) Task with goal `IsClose(Fridge) AND InHand(Apple)`. We agree with the reviewer that this example presents a challenging case, where solving the precondition `IsOpen(Fridge)` of the final goal `InHand(Apple)` undoes the subgoal `IsClose(Fridge)`, which may cause the planner to oscillate between the two subgoals. Common solutions in classical planning require considering the global planning solution to resolve the correct subgoal order [1]. However, RPN does not assume a global planning setting. Solutions within the RPN framework can be either 1) annotate the dependency IsClose(Fridge) -> InHand(Apple), which leads the planner to solve `InHand(Apple)` first regardless the state of `IsClose(Fridge)`, or 2) consider the final task goal as a subgoal block instead of individual subgoals. We will revise the paper to include a detailed discussion of this challenge.

(R1) Source of supervision. We note that the form and the granularity of the supervisions in this work are very similar to that of instructional videos [36], which are extensively used in the video understanding community. Our eventual goal is to scale RPNs to real-world tasks by learning from these instructional video datasets.

(R2) Role of learning and difference from classical symbolic planners. While RPN is inspired by classical planners, the additional assumptions and the extra information required by classical planners prohibit a fair comparison to RPN. As noted in the main text (line 5, 24, 91, 122-127), classical symbolic planners search for plans based on *planning domains*, which define high-level actions as the symbolic rules of the task domains. The main challenge for symbolic planners is thus not only to estimate entity states but even more to hand-define such rules for real-world domains at scale. The key contribution of our paper is a novel neural network architecture that learns to plan from task demonstrations without the need of a planning domain or explicit state estimation. Each component of the network directly takes high-dimensional observations as input and encapsulates a part of the overall regression planning process (as described in Sec. 4 of the paper). Furthermore, RPN works with an incomplete predicate set. For example, to plan in the kitchen domain, a classical planner would require predicates such as `Graspable` and `Activatable` to define necessary rules, whereas RPN learns to plan without these predicates (Appendix 3.2 includes a list of predicates used in the experiments). We will revise the paper to further clarify these differences and the advantages of RPN.

(R2) Application in more "realistic" settings. First, we designed our evaluation environment to emulate a realistic setting: our Kitchen 3D environment features visual scenes rendering meshes scanned from physical objects and full 6-DoF object pose variations. Second, while the chosen task domains facilitate symbolic definitions, this type of domains is widely adopted by prior works on real-world robot planning, especially in task and motion planning [12,13,19]. However, while these works assume access to symbolic planning domains, our paper presents a data-driven method that learns to perform task planning without a planning domain. Furthermore, we note that because RPN represents the planning steps with generic function approximators, we may use other representations that exhibit similar compositional structures. As noted in the paper (line 345), we plan to extend RPN to work with hybrid symbolic-geometric task goals, e.g., `On(A, B)` with a specified 6D pose. In such setting, the generated plans could be fed to an off-the-shelf motion planner to enable task and motion planning. We will update the paper to highlight the relevance.

(R4) Comparison to Rocktäschel and Riedel. We thank the reviewer for the highly relevant pointer. Rocktäschel and Riedel [2] proposes a neural theorem prover (NTP) that works with incomplete knowledge bases: it uses distributed symbolic representations to resolve non-exact symbol matches and derive new logical rules. As noted by the reviewer, NTP is related to our approach in that, similar to our regression planning mechanism, it employs a learnable backward-chaining algorithm that recursively decomposes a proving goal into subgoals, and derives new proving states until the terms in the state are supported by facts in the knowledge base. Besides the difference in the observation spaces, the main conceptual difference between NTP and RPN is that the goal of NTP is to predict if a statement is true, whereas RPN focuses on the correctness of the proof (plan) – whether the next intermediate goal in the plan is reachable and can ultimately lead to the final goal. Hence, in principle, NTP may derive the correct answer from a wrong proof, whereas a wrong plan (proof) would lead to failure in robotic tasks. On the other hand, RPN and NTP are complementary in many aspects. For example, RPN may employ a distributed symbolic representation to handle task goals specified with an open vocabulary. We will include a discussion on the relevance and will consider the notations in [2].

(R4) Details. e_t denotes the entity features at time step t. We will incorporate the comments in the next revision.

[1] D. Chapman, "Planning for conjunctive goals," *Artificial intelligence*, vol. 32, no. 3, pp. 333–377, 1987.
[2] T. Rocktäschel and S. Riedel, "End-to-end differentiable proving," in *Advances in Neural Information Processing Systems*, pp. 3788–3800, 2017.