We thank the reviewers for the reviews and suggestions, which we will incorporate into the next revised version. For brevity we denote the reviewers by **[R1]**, **[R2]**, **[R3]**, **[R4]** in the detailed responses below.

[R2][R4] Comparison with other fuzzing techniques. Our graph exploration approach is designed for tasks where we want to maximize exploration with a small interaction (number of inputs) budget. The small budget, in particular, challenges our model to learn to reason about the environment or source code. The fuzzing approaches, on the other hand, trades off test case quality with speed and scale, by using randomly (cheaply) generated test cases, and they are more suitable for cases where the exploration budget is not a limiting factor and programs do not require complex reasoning. Another significant diff. between our approach and fuzzing is that our model learns about the distribution of programs (hence we need a large dataset of programs to train on), and therefore can transfer and perform better on unseen programs; while fuzzing approaches start from scratch for each program without any cross-program adaptation.

To compare our approach with fuzzing, we adapted AFL [Zalewski] and Neuzz [She et al., 2019] to our problems. We translate all Karel programs into C programs as `afl-gcc` is required in both AFL and Neuzz. We limit the vocabulary and fuzzing strategies to provide guidance in generating valid test cases with AFL. We run AFL for 10 mins for each program, and report coverage using the test cases with distinct execution traces. Note that to get n distinct traces AFL or Neuzz may propose $N \gg n$ test cases. Neuzz is set up similarly, and initialized with the output from AFL.

# distinct inputs	2	3	5	10
joint coverage	0.63	0.67	0.76	0.81
# inputs tried	11k	31k	82k	122k

Table 1: Karel Coverage with AFL

2	3	5	10
0.64	0.69	0.74	0.77
11k	14k	17k	23k

Table 2: Karel Coverage with Neuzz

Ours	0.75
AFL	0.53
Neuzz	0.50

Table 3: Single input coverage

We report the joint/single coverage with best configs above. For comparison, our approach has a coverage of 0.75 with 1 test case and 0.95 with 5, significantly more efficient than AFL (0.76) and Neuzz (0.74). Also note that we can directly predict the inputs for new programs (in seconds), rather than taking a long time to just warm up as needed in AFL.

Using our approach on standard problems used in fuzzing is challenging. For example, the benchmark from Neuzz consists of only a few programs and its size makes it difficult to use our learning based approach that focuses on generalization across programs. On the other hand, our approach does not scale to very large scale programs yet and scalability is a challenge for future work. SMT solvers are similar to our approach in this regard as both focus on analyzing smaller functions with complex logic.

[R1][R2][R4] More related work in RL and fuzzing literature. We thank the reviewers for pointing out relevant papers, which we will properly cite in our revision. Here we briefly summarize and discuss the connections and differences to prior work. In GNN/RL related work, DOM-Q-NET [Jia et al., 2019] navigates HTML page and finishes certain tasks while the "Learning to navigate" paper [Mirowski et al., 2017] is more about handling complex visual sensory inputs. These two are like path finding, while our task is seeking optimal traversal tour (which is NP-hard). Also these two do not model the history. The Neural Graph Evolution [Wang et al., 2019] searches for the best graph structure, while we are learning to explore on a graph. The Graphical State Transitions paper [Johnson, 2017] builds a graph structure for supervised learning tasks like Q&A. Adapting them to an RL problem alone is highly non-trivial and requires significant work. For Fuzzing, we compared against an SMT solver and human experts, which are *not* weak baselines **[R2]**. The related work of Learn&Fuzz [Godefroid et al., 2017] and DeepFuzz [Liu et al., 2019] learn language models of inputs (in an unsupervised way) and sample from it to generate new inputs, which is not useful in our setting as we condition the input generation on the program being tested. AFL and Neuzz are more appropriate baselines and we report results comparing with them for testing Karel programs above.

[R1] "randomness" in RandDFS: At each state in DFS, the RandDFS picks the next unexplored action at random; while the standard DFS picks an action according to some fixed order. In random environments, these two should perform similarly. **[R1] what "# inputs generated" means:** This is the number of actions (input test cases) the agent used to explore the environment / cover the program. Note that each action corresponds to the entire generated input structure (e.g., entire maze, or string). Reward is given after generating each such input structure. **[R1] "joint coverage" in Fig 4:** It is the union of the coverage of graph nodes (program branches) using multiple generated test inputs. **[R1] using history performs better is unintuitive:** The history includes information about how the graph and coverage evolves and how we get to the current state, which might be beneficial. **[R1] what a real app traversal graph looks like:** We have provided the graphs in Fig 9 in Appendix. **[R2] transfer across tasks:** By "zero-shot" and "transfer" we mean once learned our model can be applied directly to unseen graphs (i.e. tasks) without changing the model parameters. The learning based approaches in the literature are typically trained on and only works for a single task (or program); while the fuzzing approaches do not learn and cannot adapt as effectively as our approach (see Table 1, 2 above). **[R2] "How do you initialize node embedding"** Nodes are initialized with corresponding tokens in the syntax tree for Karel/RobustFill; and constant vectors for maze exploration, as there's no features associated with them. **[R3]** Thank you for the overall positive recognition of our work. **[R4]** Thanks for the constructive feedback. We will include more discussion of related work, and the comparison with AFL and Neuzz reported above in the next revision.