Thank you for your constructive feedback. We address the reviewers' main points below; however, we will also incorporate all other feedback in the reviews into the paper's final version.

The motivation is unclear (R3, R4), in particular, as differentiable programming is not new (R4). While differentiable programming is indeed not new, in virtually all prior efforts on the topic, including the TerpreT approach that R4 mentions, one considers a *parameterized* representation of programs (essentially, an "architecture" in the language of our paper), and the learning objective is to find optimal parameters for this representation. In contrast, our approach searches over the nonparameteric space of all architectures expressed in a rich programming language. The only paper we know that searches over architectures of differentiable programs is "Houdini: Lifelong Learning as Program Synthesis" (Valkov et al., NeurIPS '18). There, architecture search is performed using type-guided enumeration and an evolutionary algorithm, i.e., two of the baselines that we beat.

The problem of searching in spaces of program architectures is also well-studied in classical program synthesis. The best-performing approaches there are based on enumeration, Monte Carlo sampling, and evolutionary algorithms (i.e., our baselines). Our paper's new observation, as noted by R1 and R2, is that we can do better than these approaches when programs are differentiable, by learning approximately admissible search heuristics and using these heuristics to guide an informed search. A version of this idea has previously come up in LASSO search (see the Related Work section); however, the idea is completely new to the program learning and differentiable programming literatures.

We do not experimentally compare against the approaches in the related work section (R4): We compare with all approaches in the related work section with which a meaningful comparison is possible. Specifically, we cannot compare against neural program induction techniques, as the outputs of these approaches are neural nets as opposed to programs. DARTS-style, gradient-based architecture search cannot be naturally extended to our setting because of the complexity of our programming language (see discussion in lines 294-299). We cannot compare against metalearning-based approaches such as "Accelerating Search-Based Program Synthesis Using Learned Probabilistic Models" (Lee et al., PLDI '18) and "DeepCoder: Learning to Write Programs" (Balog et al., ICLR '17) as well as RL-based approaches, because we do not have available a corpus of datasets and corresponding programs and must learn a program from a single dataset. We have already compared against enumerative and evolutionary architecture search, used by Valkov et al. We have also compared against Monte Carlo sampling, which underlies many of Bayesian program synthesis approaches, such as in "Sampling for Bayesian Program Synthesis" (Ellis et al., NeurIPS '16). We will be happy to compare with a MCTS baseline, as R1 recommends.

The datasets used are nonstandard/toyish (R1, R2). As R1 points out, a central focus of our work was on generating *interpretable* programs; to that end, we focused on behavior analysis applications where interpretability is an especially important concern. Prior efforts in machine learning research have used these datasets — see "Generating Multi-Agent Trajectories using Programmatic Weak Supervision" (Zhan et al., ICLR '19), "Learning Recurrent Representations for Hierarchical Behavior Modeling" (Eyjolfsdottir et al., ICML '18), "Learning fine-grained spatial models for dynamic sports play prediction" (Yue et al., ICDM '14), "Social behavior recognition in continuous video" (Burgos-Artizzu et al., CVPR '12). Also, these datasets are representative of real behavior data used by real domain scientists (sports analysts for basketball, and neuroscientists for CRIM13 and Fly-vs-Fly), and were used in real domain applications before use in machine learning research.

Clarifications on the DSL (R3). We will add further clarifying details about the DSL in the final version. We elided details of the algebraic operations and parameterized functions in the paper because these details do not affect the abstract form of our programs or the way our search algorithm works. The abstract notation for the language that we use in the paper is standard in research on Programming Languages (PL), and also appears in many prior papers in the intersection of PL and machine learning. As for our type system, it is quite basic and primarily ensures that the types of formal parameters, actual parameters, and return values are as expected; when expanding a partial architecture, we ensure that the chosen expansion is consistent w.r.t. this type system. Again, we will be happy to provide additional details in the final version if the paper is accepted.

Questions regarding the neural models used in our heuristic (R3, R4). As briefly mentioned in the paper and appendix, we use feedforward neural networks and LSTM recurrent neural networks as the neural architectures for our heuristic. The parameterization of these models varies depending on the complexity of the program using these models as relaxations. We provide the specific implementation of this in our codebase, but we will elaborate on this point in the appendix.

Other Comments. In addition to these main points, we want to thank the reviewers for their helpful suggestions on how to further improve the paper. We plan on incorporating and discussing all of these, such as a greater focus on interpretability (R2), incorporation of an MCTS baseline (R1), usage of a smaller synthetic experiment (R1), and the learning of a residual network post-program generation (R2). The detailed discussion on further clarifications or interesting future directions are deeply appreciated as well.