We thank the reviewers for their constructive feedback on our paper. We especially appreciate our reviewers' conviction that our implementation will be a widely used tool for embedding convex optimization problems in end-to-end learning models. By combining DCP, the residual approach to differentiation through a cone solver, and the proposed ASA form for parameters, we have demonstrated a general approach for efficiently differentiating through convex optimization problems and incorporating these problems as layers within standard deep learning packages. We also share the reviewers' opinions that several aspects of the papers can be improved. We address some of the main reviewers concerns and comments below, and we will edit the final version of the paper to clarify all these points.

Shared concerns. Reviewers 1 and 2 found some of our explanations of ASA form and DPP difficult to follow. We agree, based upon the reviewer comments, that this section can be clarified, and we will make the following changes.

We will expand on the definition of DCP in §3 and will revise §4 to make the definitions of ASA form and DPP more constructive. We will also explain the motivation for our ruleset (reviewer 1's guess is essentially correct). We will provide additional examples of DPP-compliant and non-compliant expressions and problems. Space permitting (or in an appendix if needed), we will also include simple figures illustrating why one expression is DPP, and why another is not.

To give a brief idea of our proposed edits, the basic idea of DPP (which we will expand upon in our revision) is this: (1) first, every parameter appearing in an expression tree as `param * expr`, where `expr` does *not* contain parameters, is cast to a constant for the purpose of DCP analysis; then (2) to be DPP, the problem must be DCP if the remaining parameters were replaced with variables. This is what we meant by our vague phrasing "jointly DCP ... [with] one important exception". As a quick example, let `lambda` be a positive parameter and `x` a variable: the expression `lambda * norm(x) + exp(lambda)` is DPP because (1) in `lambda * norm(x)`, `lambda` is cast to a constant `const_lambda` because it multiplies a parameter-free expression and (2) `const_lambda * norm(x) + exp(lambda)` is DCP in both `x` and `lambda`.

We will separately explain how to reduce certain expressions in which parameters are multiplied together (*e.g.*, `param1 * param2 * param3 * variable`) to DPP-compliant expressions. This will allow us to simplify the definition of DPP (in particular, it will allow us to delete the rules outlined in lines 139-141), without sacrificing the expressiveness of our grammar. The reduction is currently given in lines 158-162; this reduction will be described as an optional transformation that the user can perform manually (but can be easily automated), before canonicalizing a DPP program.

Reviewer 1. ASA form means that C and R must be affine. We will clarify this point. We will also change all mentions of "exact solution" to "solve to high accuracy".

We hope that our brief explanation of DPP in the previous section sheds some light on why the quadratic form in lines 147-151 is not DPP. The expression `quad_form(x, P)`, where x is a variable and P is a parameter, is not DCP in x and P (with P treated as a variable), hence it is not DPP. On the other hand, `square(norm(P*x))` is DPP, because the parameter P is first cast to a constant (since it multiplies x), and `square(norm(P*x))` is DCP (since P is treated as a constant). In the revision, we will make sure to clearly explain this.

We agree that we didn't highlight sufficiently the possible non-differentiability of the solution map. Like other layers, optimization layers can lead to undefined behavior due to non-differentiability, but, unlike other layers, it is not always obvious whether the solution map will be differentiable at a point. We strongly recommend for users to formulate their optimization layers so that the solution is unique and the derivative is well-defined, however we understand that this will not always be the case. We will add a further discussion of non-differentiability (including more specific citations) and how the library behaves in such cases. We follow the convention of recent automatic differentiation libraries (*e.g.*, PyTorch and Tensorflow), and return the gradient when it exists, and otherwise return something "reasonable." In our case we consider "reasonable" to be the least squares solution to the non-invertible linear system for the derivative.

Reviewer 2. We thank the reviewer for their appreciation of the significance of our contribution, most of all in enabling researchers to explore the applications of differentiable convex optimization layers. We will greatly clarify the exposition of ASA form and define it constructively (see shared concerns), revise the sparse canonicalization map example and Figure 1, and follow all minor suggestions.

As an alternative to multithreading, we have implemented batched computations on the GPU by providing a PyTorch callback to the conic solver SCS. Space and time permitting, we will discuss the tradeoffs of both approaches. Regardless, our open-source implementation will support both.

Reviewer 3. Our contribution is a new approach to differentiating through (smooth or nonsmooth) disciplined convex programs, considering convex programs as mappings from parameters to solution values. Computing a solution path, as in LARS, is a related but orthogonal task. Analysis of the condition number of the convex problems generated by these reductions is an interesting but orthogonal direction, and is unfortunately out of the scope of this paper.