We would like to thank the reviewers for their thoughtful advice and feedback. Reviewers raised concerns about the applicability of this work to NeurIPS. As AD is a fundamental computation required to train neural networks, perform Bayesian inference, and run many other ML algorithms, the AD tool-maker community has been well-represented at NeurIPS. In NeurIPS 2018, van Merrienboer et al survey AD and created a differentiable intermediate representation for arrays [3], de Avila Belbute-Peres et al published a differentiable physics engine [1], and in NeurIPS 2019 there was a dedicated workshop on Program Transformations for Machine Learning. This is also useful to the rest of the NeurIPS community as ML researchers rely heavily on AD tools like DSL's, PyTorch, and TensorFlow to enable their research.

Some reviewers were confused by the limitations of the approach. As we describe in lines 176-182, Enzyme is limited to producing gradients of programs whose active values have statically analyzable types and whose active functions all have statically available bitcode. Using bithacks to modify a floating-point value can result in values without statically analyzable types as it is ambiguous whether the value is a float or integer. Rather than produce incorrect code, Enzyme will emit a compile-time error if it is unable to perform an analysis needed by AD. This enables programmers to provide this information to the compiler in the form of additional attributes, a custom derivative, or other means.

Some reviewers had questions about what pieces of Enzyme's implementation are novel from other AD systems. The goal of Enzyme is to perform AD after optimization and as part of general-purpose compiler (LLVM), thereby enabling speedup over pre-optimization AD and AD on a variety of languages. This has historically not been explored as a result of the need to work on a lower-level, where information that existing tools rely upon has been lost. To remedy this, we introduce a new interprocedural type analysis (line 95) to derive type information and create a novel variant of shadow memory to support indirect function calls. Like most AD systems, we also implement well-studied (line 140) best practices such as activity analysis and caching (checkpointing), extending them to take advantage of the results of Type Analysis and LLVM's Alias Analysis for additional optimization.

A reviewer asked about how experiments were prepared. ADBench tests were directly taken from the benchmark suite, corrections and all. Additional tests were created using Tapenade's web interface or replacing programs with Adepts differentiable types (using Vector and Matrix extensions, where relevant). The two C++ tests (Euler and RK4) indeed aren't compatible with Tapenade and are shown with a red X in the plot and a N/A in the table, denoting benchmarks that don't work with that system (Figure 9).

A reviewer would have also liked to see Enzyme produce code for popular models. As part of our test suite we include a diverse set of ML techniques (LSTM, Gaussian Mixture Model, Bundle Analysis). The point of Enzyme, however, is not to replace frameworks like PyTorch or Tensorflow by better optimizing popular architectures, but to augment them by allowing ML researchers to use arbitrary existing code as part of their model without the effort of rewriting the codebase in that framework.

Some reviewers express concern that the example shown in Figure 1 of the submission could be remedied by rewriting the source code of the example. While identifying optimization opportunities is easy in small didactic examples, finding and performing such optimizations is tedious and non-obvious without automatic tools like compilers. For example, through the use of an automated tool, Doerfert et al found that "missing" function attributes in an already highly tuned DOE benchmark account for a 21% performance loss [2]. Doing so manually would require re-implementing techniques like alias analysis and code motion across thousands of lines of code. By performing AD alongside existing compiler analyses and optimizations, Enzyme can take advantage of all such optimizations automatically.

We thank the reviewers for discussing places of potential confusion. Upon further reflection, we also recognize that the paper's tone may have obfuscated certain points as well. We shall revise the writing to both clarify any confusion and generally be more objective. Finally, we would like to thank the reviewers for suggesting avenues of future research. Supporting AD of GPU and accelerator codes would definitely be valuable. While we don't explicitly address GPU's in this work, LLVM has both a frontend and backend for GPU's that should be able to integrate with Enzyme. While we've explored the specific ordering of optimizations in the experimental setup, fine-tuning the optimal place for AD in the stack remains a prime area for future work.


[1] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7178–7189. Curran Associates, Inc., 2018.

[2] Johannes Doerfert, Brian Homerding, and Hal Finkel. Performance exploration through optimistic static program annotations. In *International Conference on High Performance Computing*, pages 247–268. Springer, 2019.

[3] Bart van Merrienboer, Olivier Breuleux, Arnaud Bergeron, and Pascal Lamblin. Automatic differentiation in ml: Where we are and where we should be going. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8757–8767. Curran Associates, Inc., 2018.