**Existing Automatic Differentiation Tools**

- Differentiable DSLs (TensorFlow [1], PyTorch [2]) provide a new language where everything is differentiable. Must rewrite code, only fast if DSL matches abstractions of program.
- Operator Overloading (Adept [3], JAX [4]) tools provide differentiable versions of existing language constructs (double => adouble, np.sum => jax.sum). Often creates instruction tape to be interpreted dynamically.
- Source Rewriting (Tapenade [5]) tools statically analyze code to produce a new gradient function in the source language. Re-implements parsing/semantics => limited support for recent/complex features.

**Optimization and AD**

Regardless of approach, all existing AD tools apply on unoptimized source code. Fig 1 demonstrates how applying optimization prior to AD can be asymptotically faster than current approaches.

```c
float mag(const float* float) { //Compute magnitude in O(N)
  void norm(float out, const float* float) {
    for(int i = 0; i < N; i++) { out[i] = in[i] / mag(in); }  // LIM, then AD, O(N)
  }
  float res = mag(in);
  for(int i = 0; i < N; i++) {
    out[i] = in[i] / res;
  }
  float d_res = 0;
  for (int i = 0; i < N; i++) {
    d_res += -in[i] * in[i] / d[out[i]/res];
  }
  d[in] = d[out]/res;
}

// LI CM moves mag outside loop
  float d[O(N)]:
    for(int i = 0; i < N; i++) { out[i] = in[i] / mag(in); }  // AD then LI CM, O(N^2)
    float res = mag(in);
    for(int i = 0; i < N; i++) {
      out[i] = in[i] / res;
    }
    float d_res = 0;
    for (int i = 0; i < N; i++) {
      d_res += -in[i] * in[i] / d[out[i]/res];
    }
    d[in] = d[out]/res;
    d[mag(in, d[in], d_res)];

Figure 1. When differentiating norm, running UCM prior to AD is asymptotically faster than running AD followed by LI CM.
```

**Design**

Enzyme operates on a common compiler intermediate representation, LLVM [6], This not only enables Enzyme to operate after optimization (getting speedups like above), but also differentiate any LLVM-based language (C, C++, Fortran, Rust, Swift, Julia, Python, JaX, MLIR, PyTorch, etc).

**Evaluation**

On the CPU (Fig 5), AD after optimization is 1.2x faster than AD before optimization. On the GPU (Fig 6), AD after optimization (blue) is required for most benchmarks to run and novel AD and GPU-specific optimizations (green) provide order-of-magnitude performance improvements.

![Figure 5. Relative speedup of AD systems on ADBench+ [7] benchmarks, higher is better. A red X denotes programs that an AD system does not produce a correct gradient.](image)

![Figure 6. GPU + AD overhead with no optimizations, only LLVM optimizations (blue), and AD-specific optimizations (green). An overhead of N means computing the derivative of all inputs and original outputs is equivalent to running the original code N times.](image)

**References & Acknowledgements**

For information on installing and using Enzyme, visit enzyme.mit.edu.