MLIR: Scaling Compiler Infrastructure for Domain Specific Computation [1]

Federico Bruzzone,[1] PhD Candidate

Milan, Italy – 18 March 2026

[1]ADAPT Lab – University of Milan,
Website: federicobruzzone.github.io,
Github: github.com/FedericoBruzzone,
Email: federico.bruzzone@unimi.it
Slides: TODO

MLIR: Multi-Level Intermediate Representation

Part of the LLVM project, the MLIR is a novel approach to building **reusable** and **extensible** compiler infrastructure.

MLIR aims to address software fragmentation, improve compilation for heterogeneous hardware, significantly reduce the cost of building **domain specific compilers**, and aid in connecting existing compilers together.

Why another compiler infrastructure?

Although the *one size fits all* approach of traditional compilers (e.g., LLVM [2] or JVM [3]) has been successful for general-purpose programming, it has shown limitations in the context of domain-specific applications.

Many problems are better modeled at a **higher-** or **lower-level abstraction** — e.g., source-level static analysis of C++/Rust is difficult on LLVM IR.

Hence, many languages and frameworks developed their own intermediate representations (IRs) to leverage the **semantic information** of their domain — including TensorFlow's XLA HLO, PyTorch's Glow, Rust's MIR, Swift's SIL, Clang's CIL, and so on.

While domain-specific IRs are well-understood, their *high engineering costs* often lead to compromised infrastructure quality. This results in *suboptimal compilers* plagued by bugs, latency, and a poor debugging experience [1].

Thank You!

Bibliography

[1] C. Lattner *et al.*, “MLIR: Scaling Compiler Infrastructure for Domain Specific Computation,” in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2021, pp. 2–14. doi: 10.1109/CGO51591.2021.9370308.

[2] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis & transformation,” in *International symposium on code generation and optimization, 2004. CGO 2004.*, 2004, pp. 75–86.

[3] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, *The Java virtual machine specification*. Addison-wesley, 2013.