ML-Compiler-Bridge: Interfacing ML and Compilers

S. VenkataKeerthy\textsuperscript{1}, Siddharth Jain\textsuperscript{1},
Umesh Kalvakuntla\textsuperscript{1}, G Pranav Sai\textsuperscript{1}, Rajiv S Chitale\textsuperscript{1}, Eugene Brevdo\textsuperscript{2}, Albert Cohen\textsuperscript{2}, Mircea Trofin\textsuperscript{2}, Ramakrishna Upadrasta\textsuperscript{1}

\textsuperscript{1}IIT Hyderabad, \textsuperscript{2}Google

Eighth LLVM Performance Workshop at CGO
2\textsuperscript{nd} March 2024
ML, ML everywhere!

- Impact of ML for *hard, heuristic-based* compiler optimizations

---

**Why haven’t compilers changed?**

**Hypothesis:** They are so good, no need to change.

- Compilers extract most performance from high-level programs
- Compilers have consistently contributed to performance
- Compilers are relatively easy to create and maintain

**It is High Time to Fundamentally Redesign our Compiler Stack**

---

**Mendis’s Model of Compiler Optimization**

---

https://www.youtube.com/watch?v=w_sX9aZoZxg

---

*Compiler 2.0 (CGO’22 & LCTES’20 Keynotes) by Prof. Saman Amarasinghe*
ML, ML everywhere!

200+ works on using ML for Compiler Optimizations in the recent years!

- Ease of designing ML based Compiler Optimizations
- End-to-End Integration of ML Compiler interaction
- Transcending from Research to Deployment
ML for Compiler Optimizations

Benchmarks

Synthetic programs

Program Generation

Fuzzers

Generate/Select Programs

Choose Representations → AST/IR/PDG/…

Represent programs as vectors

S1. i = 1
S2. sum = 0
P3. while i <= 10 do
S4. read j
P5. if j <= 0 then
S6. continue
S7. sum = sum + j
P8. if sum > 100 then
S9. break
S10. i = i + 1
S11. print sum
ML for Compiler Optimizations

Represent programs as vectors

Choose ML model(s)

Perform Optimizations

// (a) Loop1
int x[N], y[N], a[N];
for (int i=1; i<N; ++i) {
  x[i+1] = x[i-1]+x[i+1];
  a[i+1] = (a[i-1]+a[i])/2.0;
}

// (b) Loop1 : Distributed
int x[N], y[N], a[N];
for (int i=1; i<N; ++i) {
  x[i+1] = x[i-1]+x[i+1];
  for (int i=1; i<N; ++i) {
    a[i+1] = (a[i-1]+a[i])/2.0;
  }
}
ML-Compiler Interaction

Input Program

Compiler

ML Models

Input serialization

Optimization Query

Output deserialization

Response

Model Output + Related Info

Training / Inference

Compilation starts

Opt 1

Materialize Predictions

Opt n

Input deserialization

Output serialization
ML-Compiler Interaction

Input Program -> Compiler

Compilation starts

Opt 1

Materialize Predictions

Opt n

Compiler

Input serialization -> Optimization Query

Output deserialization

Response

Opt n

Input deserialization

Output serialization

ML Models

Training / Inference

Querying

Opt 1

Optimizations

Model Output + Related Info

Opt n
ML-Compiler Interaction

- **Highly Important**
  - Scalability
  - Compile Time Issues
  - Memory Issues
  - ...

- **Determines the practicality**
  - Deployment
  - Usability
  - ...

Materialize Predictions

Compilation starts

Training / Inference

Querying
ML-Compiler Interaction

Current Approaches

- No single standard approach
  - Python wrappers, Compiler flags

- No deeper integration
  - High-level interfacing

- Model written with C++ APIs
  - Tight coupling of APIs
# Current Limitations

## Scalability
- Python/C++ wrappers
- 6x – 100x slowdown

## Integration
- Not all outputs can be communicated via flags
- Register Allocation, Instruction Scheduling, ...

## Programmability
- Models written in C++ are not ML developer friendly
- RLLib, SciPy, ...

## Portability
- Support for diverse ML frameworks
- TF, PyTorch, JAX, ...
Current Limitations

m x n problem 😞
Our Proposal
ML-Compiler-Bridge

Compilers
- LLVM
- GCC
- MLIR
- Pluto

Data-In
ML-Compiler-Bridge
SerDes
MLModelRunner
Inter-process Model Runners
In-process Model Runners
Data-Out

ML Models
- PyTorch
- Keras
- TensorFlow
- JAX
- CompilerGym
- RLLib
- PolyGym
- Stable-baselines3
- OpenAI Gym

Data-In
Data-Out
ML-Compiler-Bridge

Compilers
- LLVM
- GCC
- MLIR
- Pluto

Data-In

Data-Out

ML-Compiler-Bridge

SerDes
- ProtobufSerDes
- JSONSerDes
- BitstreamSerDes

Inter-process
- gRPCModelRunner
- pipeModelRunner

In-process
- ONNXModelRunner
- TFModelRunner

Data-In

Data-Out

ML Models
- PyTorch
- Keras
- TensorFlow
- JAX
- CompilerGym
- RLlib
- PolyGym
- OpenAI Gym
- Stable-baselines3
Model Runners: Medium of Communication

Two Broad Model Runners

Inter-Process Model Runners
- Compiler and the ML model runs as two concurrent processes.
  - gRPC
  - Unix-style Named Pipes

In-Process Model Runners
- ML model is part of the compiler
  - ONNX C++ Runtime
  - TensorFlow AOT model

Designed for **Training**

Designed for **Inference**
Inter-process Model Runners: gRPC

Proto File 1
::
Proto File N

<name>.proto

Auto Generated Header Files

gRPCService 1
::
gRPCService N

Compiler Pass 1

ML Module 1

Compiler Pass N

ML Module N

libgRPCModelRunner.a

gRPC ModelRunner
gRPC Model Runner - proto description

```proto
syntax = "proto3";
// Package name for current optimization
package helloMLBridgegRPC;
// Service class RPC declarations
service HelloMLBridgeService {
  // RPC to query compiler
  rpc queryCompiler(ActionRequest) returns (TensorResponse) {} 
  // RPC to get Advice from model
  rpc getAdvice(TensorResponse) returns (ActionRequest) {} 
}
// Data structures for request and response messages
message TensorResponse { repeated float tensor = 1; }
message ActionRequest { int32 action = 1; }
```

RPC Service Class

RPC to Query Compiler

RPC to Query Model

Request/Response Data Structure
Inter-process Model Runners: Pipes
Pipe Model Runner - Internals

```cpp
... void *PipeModelRunner::evaluateUntyped() override {
    auto *data = SerDes->getSerializedData();
    send(data);
    auto *reply = receive();
    return SerDes->deserializeUntyped(reply);
}
...```

- Send data on pipe
- Get serialized data from SerDes
- Read received data on pipe (Blocking)
- Deserialized received data and return
In-process Model Runners: TensorFlow AOT

TF model → TF CLI → Building LLVM → Querying Model → Model Output

- Compile AOT model (.o +.h files)
- Build LLVM with AOT model
- Query model as function calls
In-process Model Runners: ONNX

ONNX - Framework neutral, interoperable infrastructure for trained model integration

SerDes: Serialization-Deserialization Module

Data to send

BitStream

Json

Protobuf

message Data {
  repeated float embedding = 1;
}

length(data)

send(hdr+data)

{  
  'name': '...',
  'type': 'float',
  'shape': [1, 300]
}

Protobuf BitStream

Json

Protobuf

message Data {
  repeated float embedding = 1;
}

length(data)

send(hdr+data)

{  
  'name': '...',
  'type': 'float',
  'shape': [1, 300]
}
## Comparison of Different Model Runners

<table>
<thead>
<tr>
<th></th>
<th>gRPC</th>
<th>Pipes</th>
<th>ONNX</th>
<th>TF-AOT</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Multithreaded</strong></td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td><strong>Compilation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Distributed</strong></td>
<td>✔</td>
<td>✗</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Training</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Single process</strong></td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>(Model is part of the compiler)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Auto-serialization</strong></td>
<td>✔</td>
<td>✔</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td><strong>Communication</strong></td>
<td>✗</td>
<td>✗</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td><strong>Robustness</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ML Framework</strong></td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td><strong>Independent</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
#include "MLCompilerBridge/MLModelRunner.h"
#include "MLCompilerBridge/yourMLModelRunner.h"

// Instantiate the required model runner with SerDes type
MLModelRunner *MLRunner = std::make_unique<yourModelRunner>(Arg,
    SerDes::Kind::yourSerDesType);

// Process Input Features
std::pair<std::string, InType> p = ... // Input
MLRunner->populateFeatures(p);

// Get ML Advice/Output
OutType advice = MLRunner->evaluate<OutType>();

// Use the obtained advice
...
import CompilerInterface as CI

# Instantiate the required CompilerInterface with serdes type
interface = CI.YourCompilerInterface(Arg, yourSerdesType)

while True:
    ...
    # Populates buffer with advice
    interface.populate_buffer(advice)
    # Send buffer data to compiler and wait for next request
    response = interface.evaluate()
    ...
    # Break on condition

Populating buffer with advice data
Creating CompilerInterface Instance
Responding to compiler with advice
# Multi-Language Support: C

```c
#include "MLModelRunner/C/ONNXModelRunner.h"
#include "MLModelRunner/C/PipeModelRunner.h"

// Instantiate the required model runner with SerDes type
PipeModelRunnerWrapper *pmr = createPipeModelRunner("plutopipe.out", "plutopipe.in", config);

// Process Input Features
float *features = ... // Input
populateFloatFeatures(pmr, "tensor", features, n);

// Get ML Advice/Output
int advice = evaluateIntFeatures(pmr);
// Use the obtained advice
...
```
Adding New Model Runners + SerDes

```cpp
#include "MLModelRunner/MLModelRunner.h"

class NewModelRunner : public MLModelRunner {
public:
    // Custom ModelRunner Constructor
    NewModelRunner();
    virtual ~NewModelRunner();
private:
    // Function to establish communication
    void *evaluateUntyped() override;
    // Functions to send and receive data
    void send(void *data);
    void *receive();
};

#include "SerDes/baseSerDes.h"

class NewSerDes : public BaseSerDes {
public:
    NewSerDes() : BaseSerDes(BaseSerDes::Kind::NewSD){};
    void setFeature(const std::string name, const int value) override;
    void setFeature(const std::string name, const float value) override;
    ...;
    void *getSerializedData() override;
    void cleanDataStructures() override;
private:
    void *deserializeUntyped(void *data) override;
};
```
Supports Wider Use-Cases...

**RL4ReAl - Register Allocation**
- Communication: gRPC based multiple times
- Agents: Multiple hierarchical agents
- Model Type: PyTorch (GNN + FCNN)
- Model Input: Interference graph + node embedding
- Model Output: Colour map

**POSET-RL - Phase Ordering**
- Communication: Opt flag based multiple times
- Agent: Single agent
- Model Type: PyTorch (FCNN)
- Model Input: IR2Vec vectors
- Model Output: Pass sequence


Jain, et al., POSET-RL: Phase ordering for Optimizing Size and Execution Time using Reinforcement Learning. ISPASS 2022
Supports Wider Use-Cases...

- **Communication**: Python Wrapper based once at end
- **Agents**: Multiple agents
- **Mode Type**: PyTorch (GNN + FCNN)
- **Model Input**: IR2Vec vectors
- **Model Output**: Distribution sequence


---

- **Communication**: Precompiled TF model
- **Agents**: Single agent
- **Mode Type**: TensorFlow (FCNN)
- **Model Input**: Feature vector
- **Model Output**: Binary (yes/no)

Training Time Improvements

POSET-RL Training Time Comparison

RL4ReAI Multi-worker Training Time Comparison
Compile (Inference) Time Improvements: POSET-RL
Support for Multi-threaded Compilation

![Graph of Compile Time (s) for Inliner and RL4Real with varying threads](image)

- **Inliner**
  - 1 Thread: 800 seconds
  - 2 Threads: 600 seconds
  - 4 Threads: 400 seconds
  - 8 Threads: 200 seconds

- **RL4Real**
  - gRPC: 6000 seconds
  - 1 Thread: 5500 seconds
  - 2 Threads: 5000 seconds
  - 4 Threads: 4500 seconds
  - 8 Threads: 4000 seconds
ML-Compiler-Bridge with CompilerGym

- Inference time comparison with CompilerGym’s phase ordering model
- Model exported as ONNX model and queried using ONNXModelRunner

Performance of Individual Model Runners

Round Trip-Times (RTT) for querying model with data of different lengths
Support for MLIR & Pluto

**MLIR**

RTT for querying model with data of different lengths
Summary - ML-Compiler-Bridge

- **Scalable, Lightweight suite of model runners and serializers**
  - Supports Multiple Languages
  - Compiler and ML-Framework Independent
  - Supports deeper and high-level interfacing with compilers

- **Plug-and-Play approach** for ML based Compiler Optimizations

- Easier transition from research to deployment

- We plan to upstream relevant portions to LLVM in addition to what is available
Thank You!

S. VenkataKeerthy | Siddharth Jain

Interested? Please get in touch with us
Visit our Poster @ C4ML (1800 hrs, Reception Area)

Looking for Extensions and Contributions

https://compilers.cse.iith.ac.in/research/mlcompilerbridge