ML-LLVM-Tools: Towards Seamless Integration of Machine Learning in Compiler Optimizations

Siddharth Jain[1], **S. VenkataKeerthy**[1] , Umesh Kalvakuntla[1],

Albert Cohen[2], Ramakrishna Upadrasta[1]

IIT Hyderabad[1], Google[2]

భారతీయ సాంకేతిక విజ్ఞాన సంస్థ హైదరాబాద్
भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

European LLVM Developers' Meeting
10th May 2023

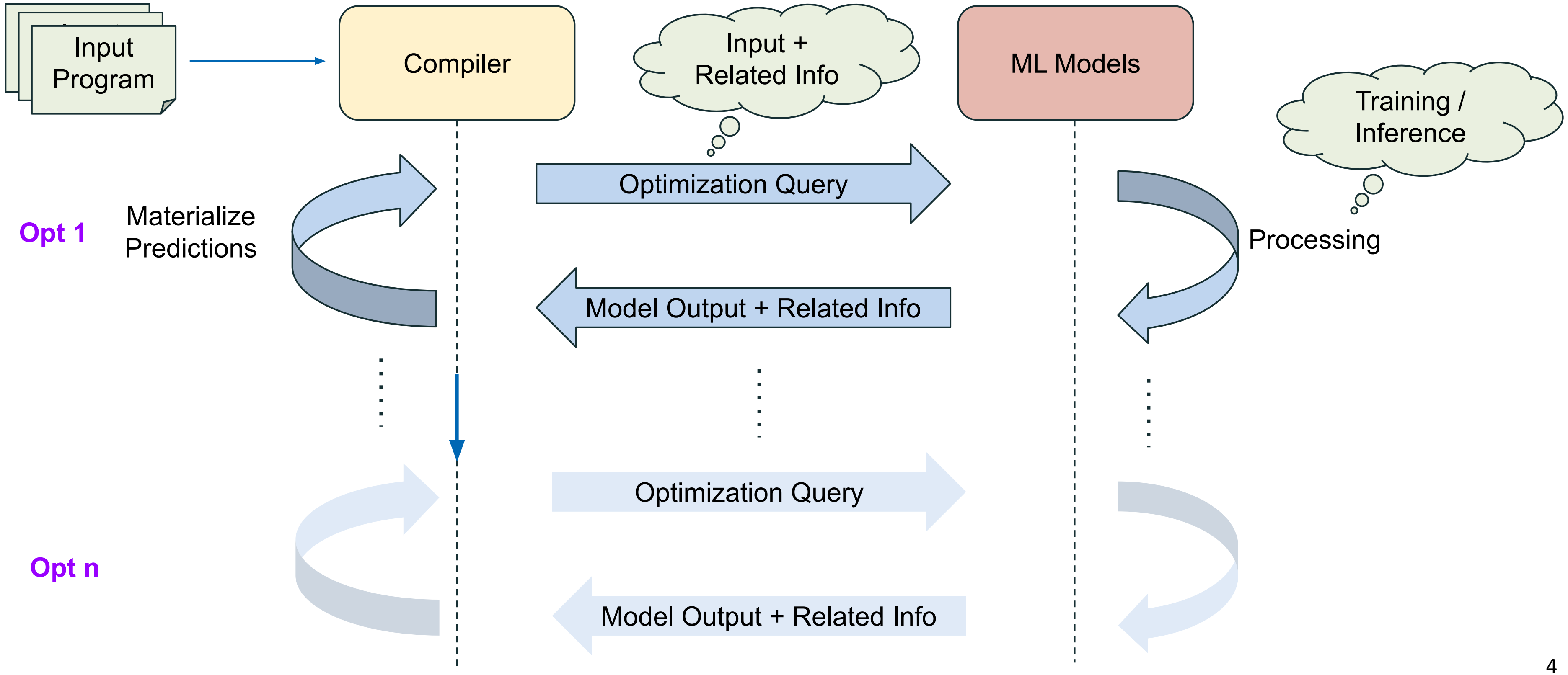
Overview

- ML in Compiler Optimizations
  - Scheme of ML in Compiler Optimizations
  - Proposed Infrastructure
- LLVM-gRPC: gRPC based framework to support Training
  - LLVM-gRPC Usage
  - Use Case: RL4ReAl, IR2Vec
- LLVM-InferenceEngine: ONNX based framework to support Inference
  - Proposed Inference Flow
  - LLVM-InferenceEngine Usage
  - Compile Time Comparison
- Related Works
- Summary

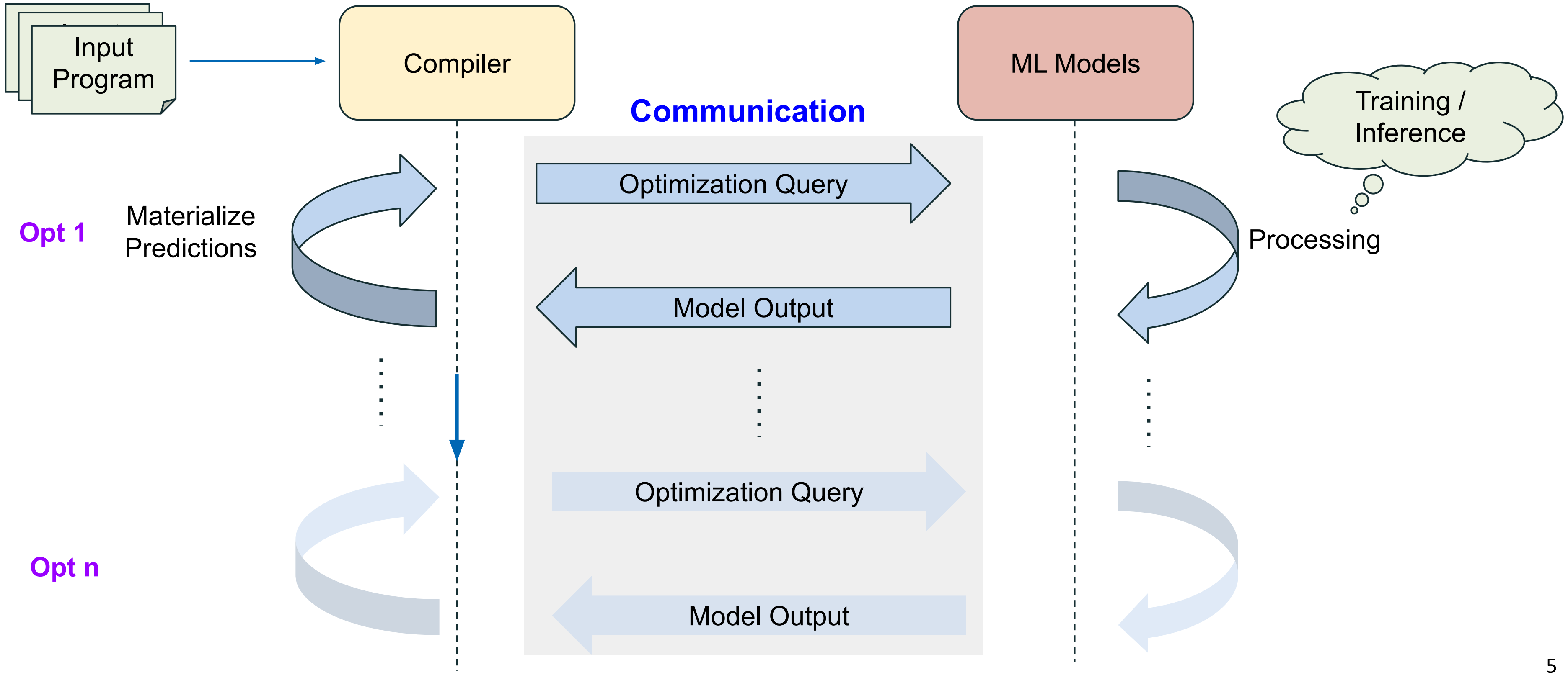
ML in Compiler Optimizations

- Impact of ML in *hard, heuristic-based* compiler optimizations
  - Success of ML in NLP, Image Processing, etc.
- Several ML based compiler optimizations exist
  - From late 90s to date
- ML based optimizations
  - Loop Vectorization, Loop Distribution, Function Inlining, Phase Ordering, Register Allocation, …
- ML in LLVM
  - Inlining decisions (From 11.x), Eviction in Register Allocation (From 14.x)

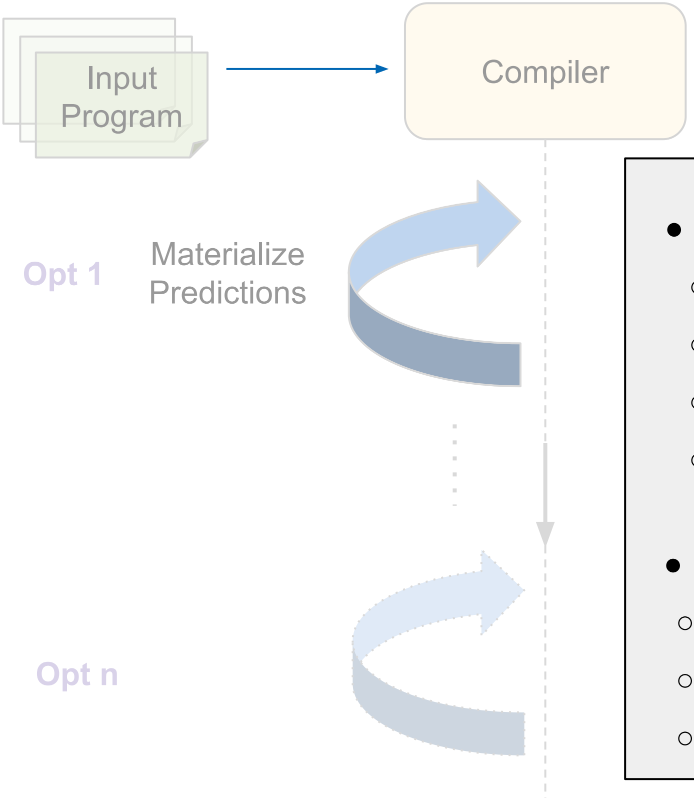
Scheme of ML in Compiler Optimizations



Scheme of ML in Compiler Optimizations

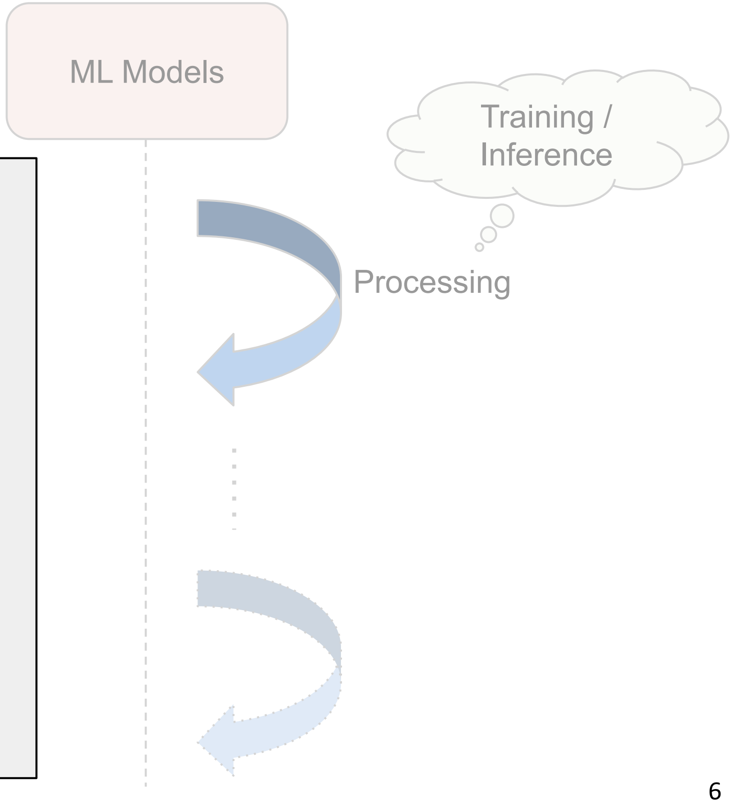


Focusing on Communication ...

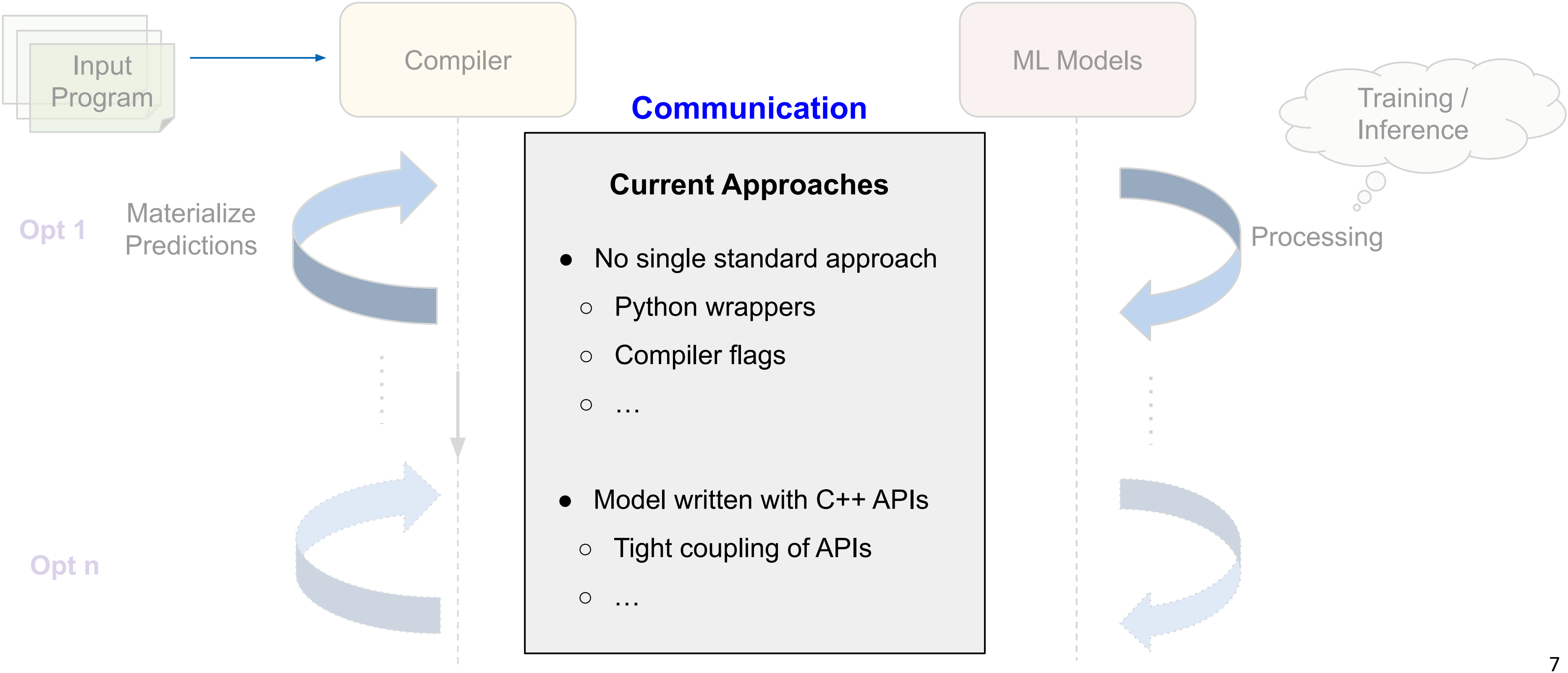


Communication

- **Highly Important**
  - Scalability
  - Compile Time Issues
  - Memory Issues
  - …
- **Determines the practicality**
  - Deployment
  - Usability
  - …



Focusing on Communication …



Current Approaches

- No single standard approach
  - Python wrappers
  - Compiler flags
  - …
- Model written with C++ APIs
  - Tight coupling of APIs
  - …

Limitations of Current Approaches

Scalability	Integratability	Programmability	Portability
• Python/C++ wrappers • 6x – 100x slowdown **Phase Ordering, Loop Distribution, …**	Not all outputs can be communicated via flags **Register Allocation, Instruction Scheduling, …**	Models written in C++ are not ML developer friendly **RLLib, SciPy, …**	Support for diverse ML frameworks **TF, PyTorch, JAX, …**

Need for **scalable**, **versatile** and **common** framework for ML-based optimizations in LLVM

Proposed Infrastructure

Framework + Architecture independent Infrastructure in LLVM

Training

- ML model development in any generic framework
- ML practitioners can develop solutions in Python

LLVM-gRPC

gRPC based library

Inference

- Within LLVM
- Trained models to be exported and linked with LLVM toolchain

LLVM-InferenceEngine

ONNX based library

LLVM-gRPC

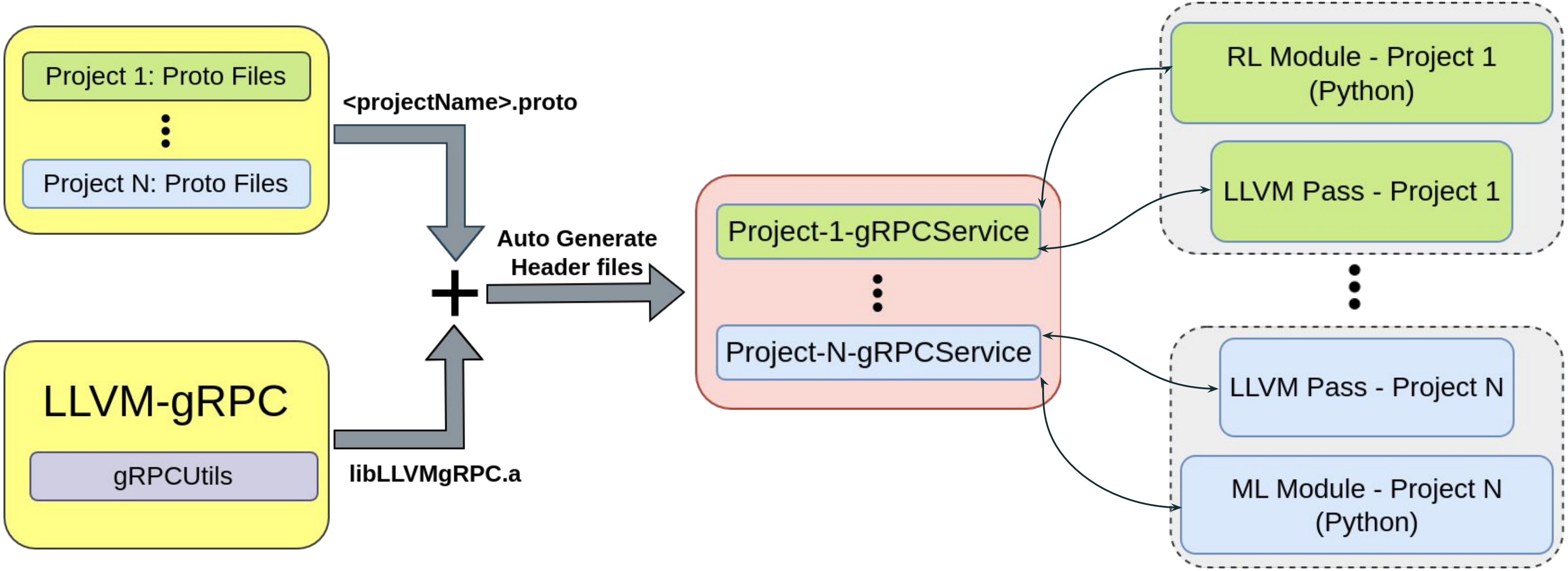
gRPC based framework to support Training

LLVM-gRPC

- Need for a seamless connection between LLVM and Python ML workloads
    - Interprocess communication
- gRPC: Modern open source high performance Remote Procedure Call
- LLVM-gRPC
    - Works as an LLVM library
    - Easy integration – As simple as implementing a few API calls
    - Support for any ML + RL workloads
- Use-case: RL4ReAl [CC'23]

S. VenkataKeerthy, Siddharth Jain, Anilava Kundu, Rohit Aggarwal, Albert Cohen, and Ramakrishna Upadrasta. RL4ReAl: Reinforcement Learning for Register Allocation. CC 2023. https://compilers.cse.iith.ac.in/publications/rl4real/

LLVM-gRPC + Passes



Example Proto File

```
syntax = "proto3";

package demopass;

// demo pass Service
service demoPass{
// RPC function to send and receive data
// between server and client
rpc getPassInfo(RequestData) returns (PassInfo) {}
}
```

demoPass: Defines the service (a C++ class) which will be auto generated

getPassInfo: Defines the RPC function which has to be overridden

```
message RequestData {
string functionName=1;
}

message PassInfo {
int32 numInstruction=1;
}
```

Datafields that will be auto generated using gRPC

Files generated on compiling proto file:

- demoPass.grpc.pb.cc
- demoPass.grpc.pb.h
- demoPass.pb.cc
- demoPass.pb.h
- demoPass_pb2_grpc.py
- demoPass_pb2.py

LLVM-gRPC Usage: C++ Server

```
using demopass::RequestData;
using demopass::PassInfo;
using grpc::Status;
using grpc::ServerContext;
using demopass::demoPass;
```

Types coming from demoPass.grpc.pb.h

```
struct Hello : public FunctionPass,demoPass::Service,gRPCUtil {
```

Inheriting classes

```
grpc::Status getPassInfo(grpc::ServerContext* context,
  const RequestData* request, PassInfo* response) override {
   // Pass logic to handle the request goes here
   ...
   return Status::OK;
}
```

Implementation of gRPC function

```
bool runOnFunction(Function &F) override {
   ...
   RunService(this,"0.0.0.0:50051");
   ...
   if(exit_requested) { free(exit_requested);}
   return false;
}
}
```

Blocking call to start the C++ server

Exiting blocking call

LLVM-gRPC Usage: Python Client

```
import grpc
import demoPass_pb2_grpc, demoPass_pb2
```

Service class defined in demoPass_pb2_grpc.py

```
class demoPassClient(demoPass_pb2_grpc.demoPassServicer):
```

Inheriting classes

```
  def __init__(self):
    self.host='localhost'
    self.server_port=50051
    self.channel=grpc.insecure_channel(
     '{}:{}'.format(self.host,self.server_port))
    self.stub= demoPass_pb2_grpc.demoPassStub(self.channel)
```

Creating a channel and stub from existing service

```
  def getRequest(self,requestData):
    request=demoPass_pb2.RequestData(requestData)
    return  (self.stub.getPassInfo(request))
```

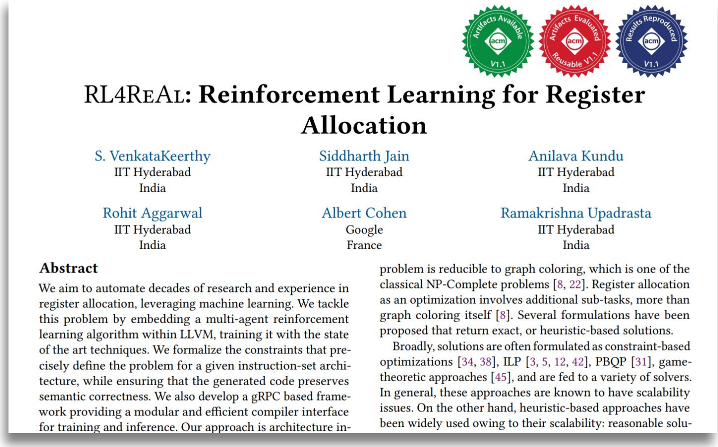
Call to gRPC function

```
if __name__ == '__main__':
  client=demoPassClient()
  functionName=demoPass_pb2.functionName(self.current_fuction_name)
  instruction_count=client.getPassInfo(functionName)
```

Use Case: RL4ReAl

RL4ReAl: Reinforcement Learning for Register Allocation

- RL based register allocator for LLVM compiler
- Models regalloc as graph coloring problem
- Based on MIR2Vec for Machine IR
    - An Extension of IR2Vec
- Uses LLVM-gRPC



RL4REAL: **Reinforcement Learning for Register Allocation**

S. VenkataKeerthy, IIT Hyderabad, India
Siddharth Jain, IIT Hyderabad, India
Anilava Kundu, IIT Hyderabad, India
Rohit Aggarwal, IIT Hyderabad, India
Albert Cohen, Google, France
Ramakrishna Upadrasta, IIT Hyderabad, India

Abstract

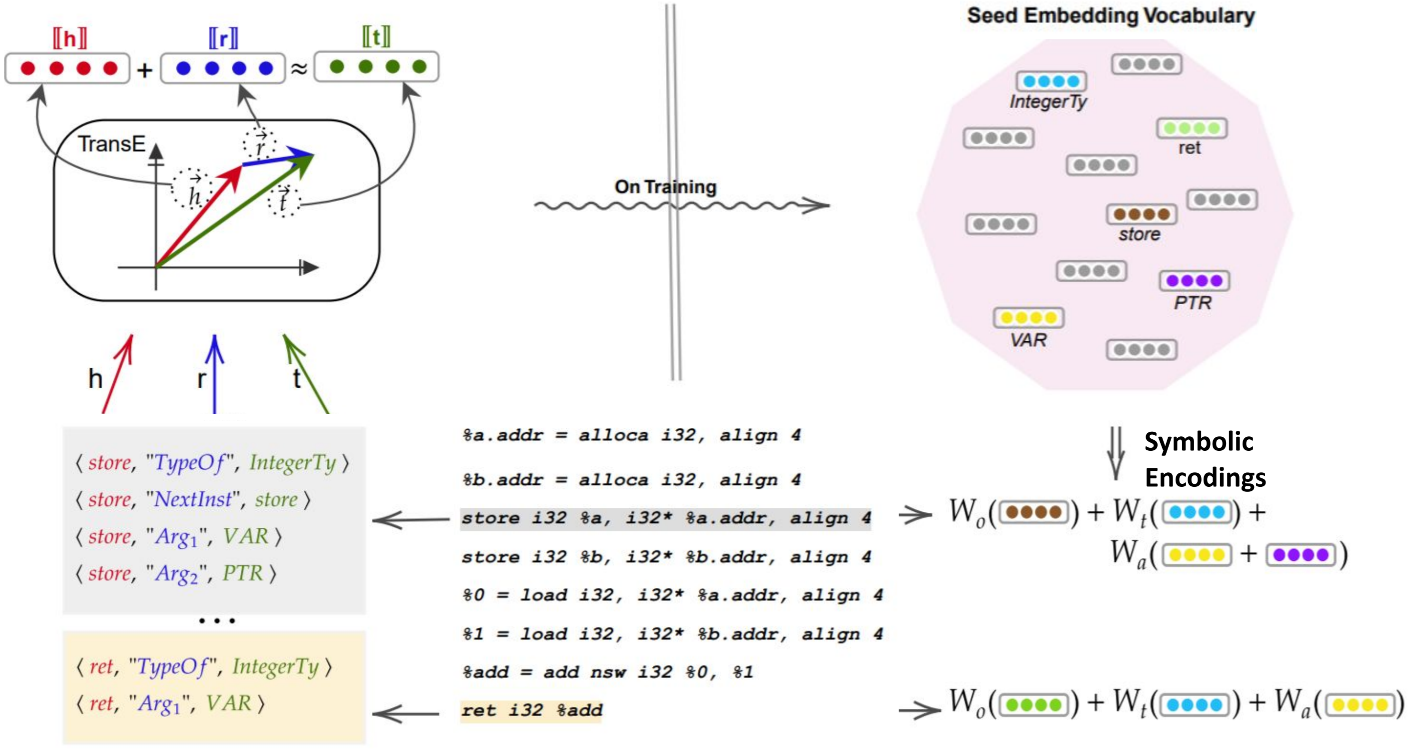
We aim to automate decades of research and experience in register allocation, leveraging machine learning. We tackle this problem by embedding a multi-agent reinforcement learning algorithm within LLVM, training it with the state of the art techniques. We formalize the constraints that precisely define the problem for a given instruction-set architecture, while ensuring that the generated code preserves semantic correctness. We also develop a gRPC based framework providing a modular and efficient compiler interface for training and inference. Our approach is architecture in-

problem is reducible to graph coloring, which is one of the classical NP-Complete problems [8, 22]. Register allocation as an optimization involves additional sub-tasks, more than graph coloring itself [8]. Several formulations have been proposed that return exact, or heuristic-based solutions.

Broadly, solutions are often formulated as constraint-based optimizations [34, 38], ILP [3, 5, 12, 42], PBQP [31], game-theoretic approaches [45], and are fed to a variety of solvers. In general, these approaches are known to have scalability issues. On the other hand, heuristic-based approaches have been widely used owing to their scalability: reasonable solu-

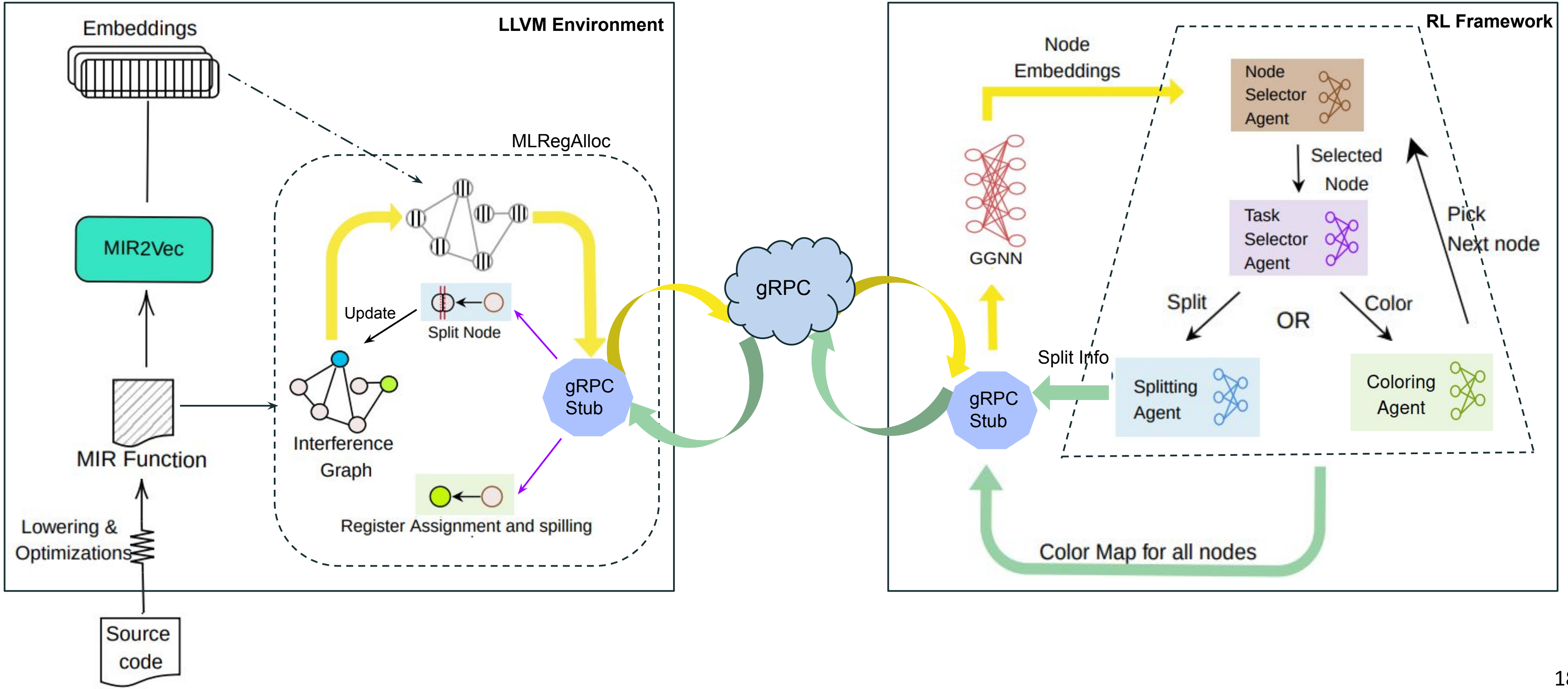
S. VenkataKeerthy, Siddharth Jain, Anilava Kundu, Rohit Aggarwal, Albert Cohen, and Ramakrishna Upadrasta. RL4ReAl: Reinforcement Learning for Register Allocation. CC 2023. https://compilers.cse.iith.ac.in/publications/rl4real/

IR2Vec: LLVM IR Based Scalable Program Embeddings



S. VenkataKeerthy, Rohit Aggarwal, Shalini Jain, Maunendra Sankar Desarkar, Ramakrishna Upadrasta, and Y. N. Srikant. IR2VEC: LLVM IR Based Scalable Program Embeddings. ACM TACO. 2020. https://compilers.cse.iith.ac.in/projects/ir2vec/

RL4ReAl: Reinforcement Learning for Register Allocation



Pros + Cons ...

Ease of development

- Transparent to DL/RL algorithms/policies
- Supports diverse ML/DL frameworks

Ease of training

- Multiple GPUs distributed training
- Parallel workers for sample collection

Reusability of code

Write specifications once and use in both Python and C++

However, LLVM-gRPC is insufficient for inference -

- Overhead on compile time
  - Interprocess communication
- Not transparent to user (application developer)

LLVM-InferenceEngine

ONNX based framework to support Inference

LLVM-InferenceEngine

- Framework neutral, interoperable infrastructure for trained model integration
- ONNX: Open Neural Network Exchange
  - Linux Foundation Project (LF AI & Data)
  - Operates in most of the native languages
  - Supported by all major ML/DL frameworks
- Usecase study - RL4ReAl [CC'23], POSET-RL [ISPASS'22]

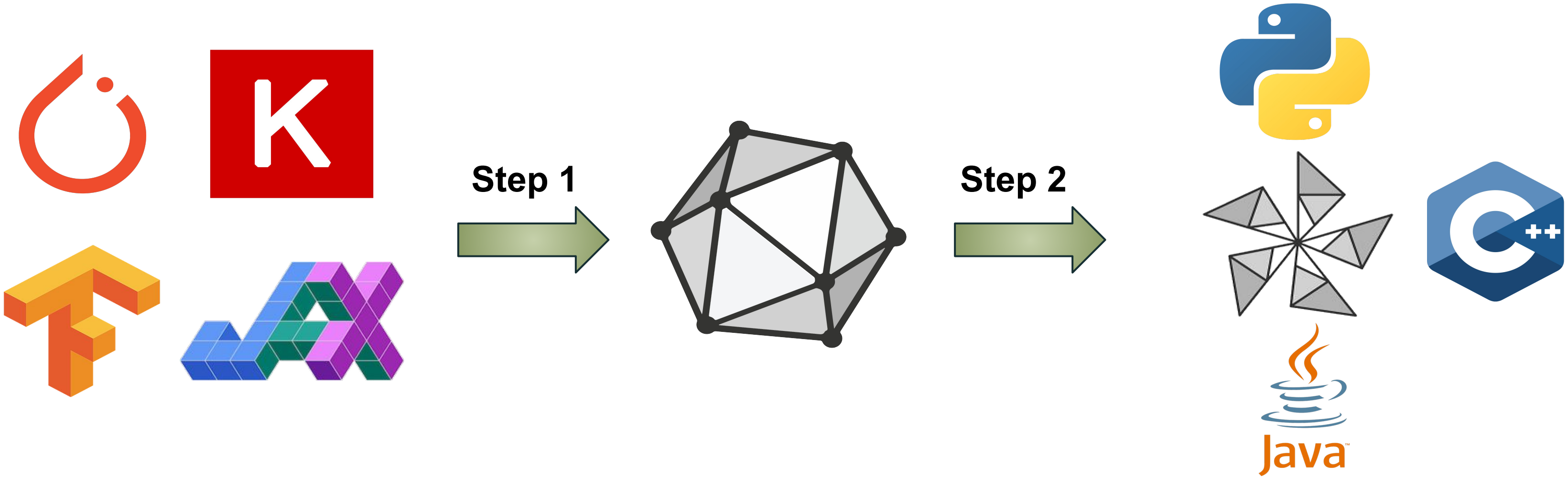
ONNX

ONNX. Open Neural Network Exchange. 2017, https://github.com/onnx/onnx

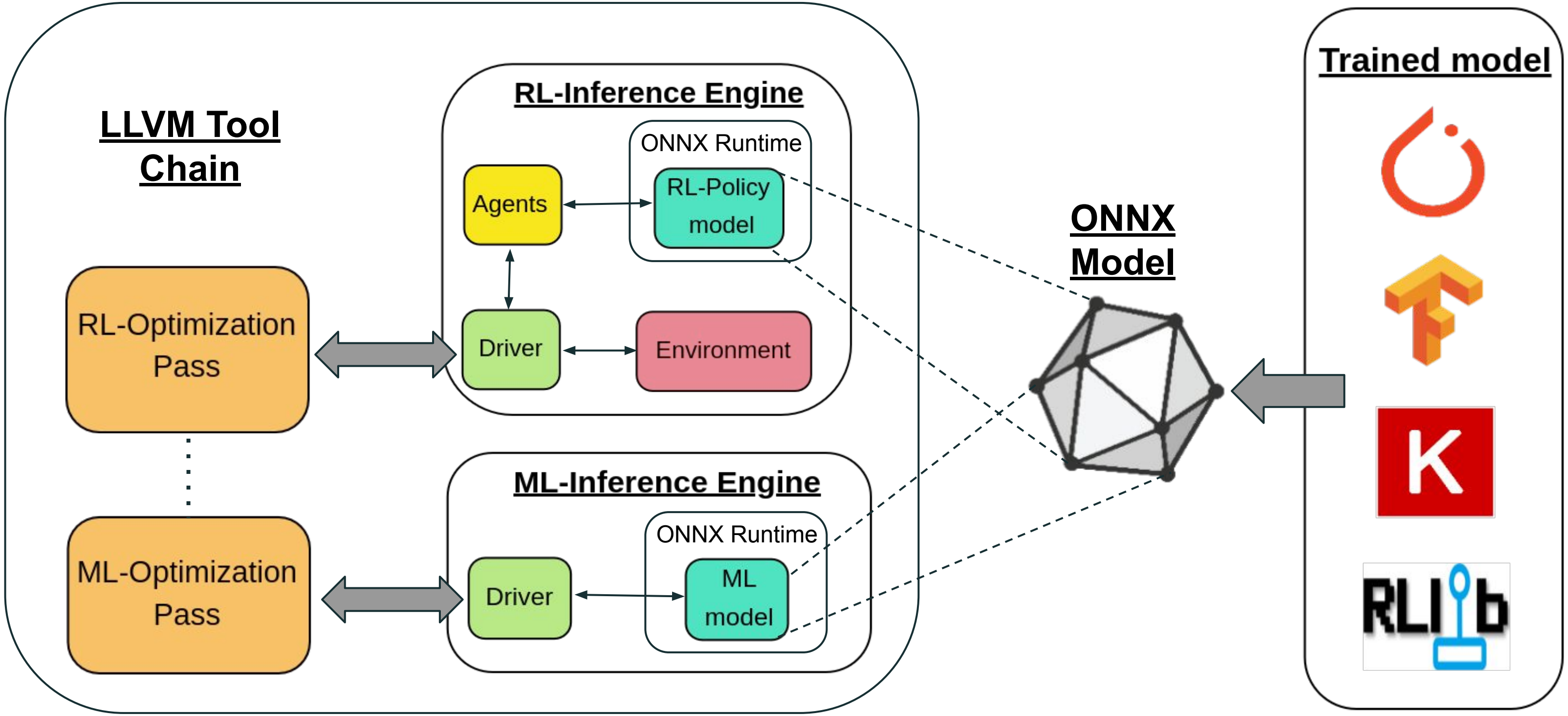
Shalini Jain, Yashas Andaluri, S. VenkataKeerthy and Ramakrishna Upadrasta. POSET-RL: Phase ordering for Optimizing Size and Execution Time using Reinforcement Learning. ISPASS 2022. https://compilers.cse.iith.ac.in/projects/posetrl/

Model Integration

- **Step 1:** Exporting trained model from native to ONNX format
- **Step 2:** Importing model in compiler with ONNX (C++) runtime environment



Proposed Inference Flow



LLVM-InferenceEngine Usage

```
#include "environment.h"
#include "inference-engine.h"

struct Hello : public FunctionPass, Environment {

bool runOnFunction(Function &F) override {
  ...

 InferenceEngine* inference_driver =
     new InferenceEngine(Environment* env);
 inference_driver->getPassInfo(PassData passData,
     OptInfo &predictions);

  ...
}
```

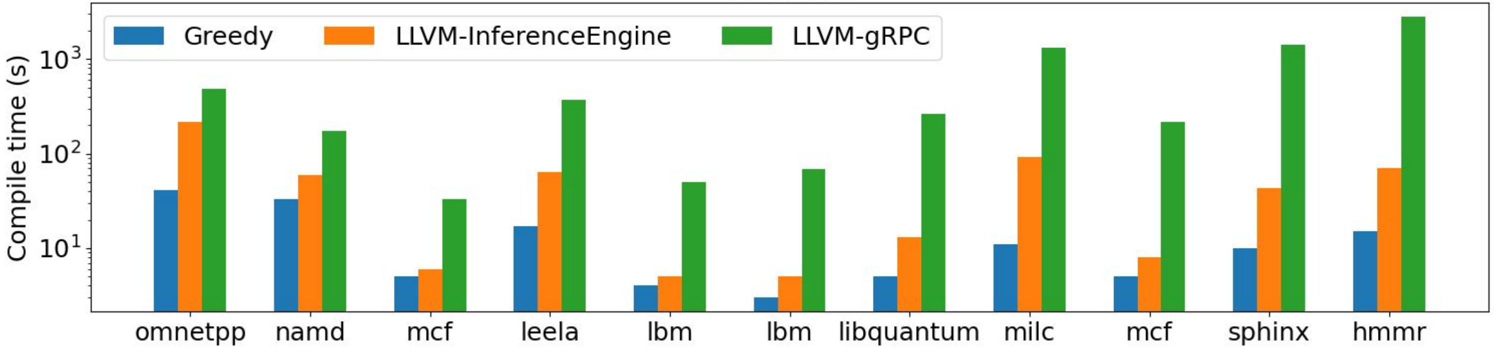
Inheriting Environment class

InferenceEngine: Creates instance of class InferenceEngine class

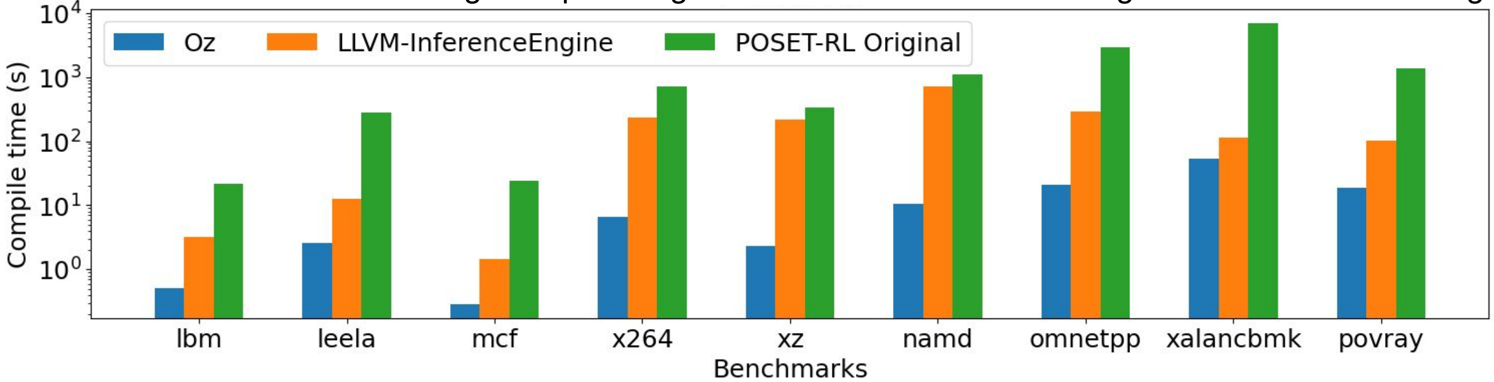
getPassInfo: Function to compute predictions from model

Compile Time Comparison

RL4ReAl: Reinforcement Learning for Register Allocation



POSET-RL: Phase ordering for Optimizing Size and Execution Time using Reinforcement Learning



Advantages

Features/Advantages of LLVM-InferenceEngine

In-process communication

- No RPC calls, IO, etc.

Lesser compilation time overhead

- No communication overhead

Versatile + Common infrastructure

- Framework and model agnostic

Transparent to the user/programmer

Other Related Works

- MLGO: A Machine Learning Framework for Compiler Optimization
    - Integrated with LLVM
    - Uses TensorFlow APIs and/or raw inter-process communication
    - We would like to explore different scenarios and use cases
        - RL Vs. ML, …; Single Vs. Multiple communication
- CompilerGym
    - Provides environments for training RL based compiler optimizations

Mircea Trofin, et al. "MLGO: a machine learning guided compiler optimizations framework." arXiv preprint 2021. https://arxiv.org/abs/2101.04808

Chris Cummins, et al. "CompilerGym: Robust, Performant Compiler Optimization Environments for AI Research." CGO 2022. https://github.com/facebookresearch/CompilerGym

Summary

- Scalable, Versatile and Common framework for ML-based optimizations in LLVM
  - Framework + Architecture independent Infrastructure
- Two components
  - Training - LLVM-gRPC
  - Inference - LLVM-InferenceEngine
- gRPC based training within Python in a framework independent manner
- In-memory ONNX based library for inference in a transparent manner
- Infrastructure is lightweight showing promising trends
- https://compilers.cse.iith.ac.in/publications/ml-llvm-tools

Thank you!

https://compilers.cse.iith.ac.in/publications/ml-llvm-tools