# RLgraph: Flexible Computation Graphs for Deep Reinforcement Learning

MICHAEL SCHAARSCHMIDT | SVEN MIKA | KAI FRICKE | EIKO YONEKI

PRESENTATION BY GRZEGORZ WILK

# Background

Standardization of high-level graph frameworks is a big driving force behind the success of ML.

Yet, it is happening very slowly for reinforcement learning settings.

Examples include:

- OpenAI
- TensorForce
- Ray RLlib

# What is reinforcement learning?

Instead of a supervised setting, where all the training data is available in advance,

Reinforcement learning revolves around defining a reward system for an agent and letting it freely produce strategies.

# Separation of Concerns

- Managing trial runs in the testing environment
- Managing execution environment
  - Distributed coordination
  - Device strategies e.g. GPU utilization
- Driving the learning models
  - Logical component composition
  - Backend graph definition for specific frameworks

# What does separation of logic achieve?

- Encourage code reuse
- Speed up iterative development
- Allow for sub-component testing
- Allow for a higher-level reasoning about the components
- Allow for identification of optimizations

# Major contribution

Meta-graph.

This provides an abstraction that allows to compose various components of the reinforcement learning procedure and re-compose them in different ways.

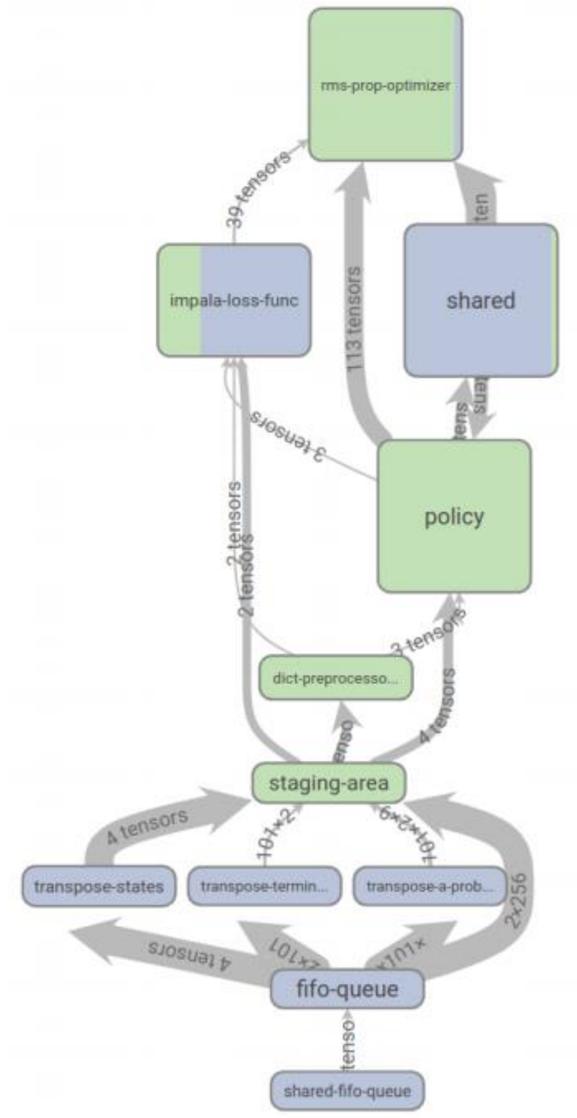



*Figure 9.* TensorBoard visualization of RLgraph's IMPALA learner. As all operations and variables are organized in components under separate scopes, dataflow between components is clear.

# How is the meta graph created?

Three stages:

- ➢Composing the components
- ➢Traversing all the call paths to provide a skeleton of the meta-graph
- ➢Building the backend-specific implementation, by allocating required data-structures and defining operations

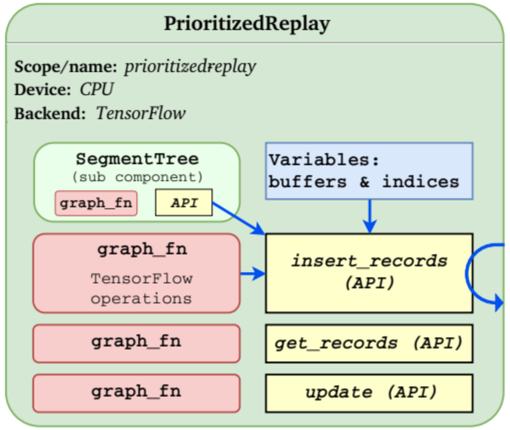



*Figure 2.* Example memory component with three API methods.

- ➢* The framework also supports a define-by-run execution mode, where the third phase isn't pre-build, instead dynamically created according to the shape of data passed around.

# Conveniences provided

- Ability to test only parts of the meta-graph
- Convenient nesting, merging, splitting and folding of data passed around
- Shape and type checking

# Backend independence

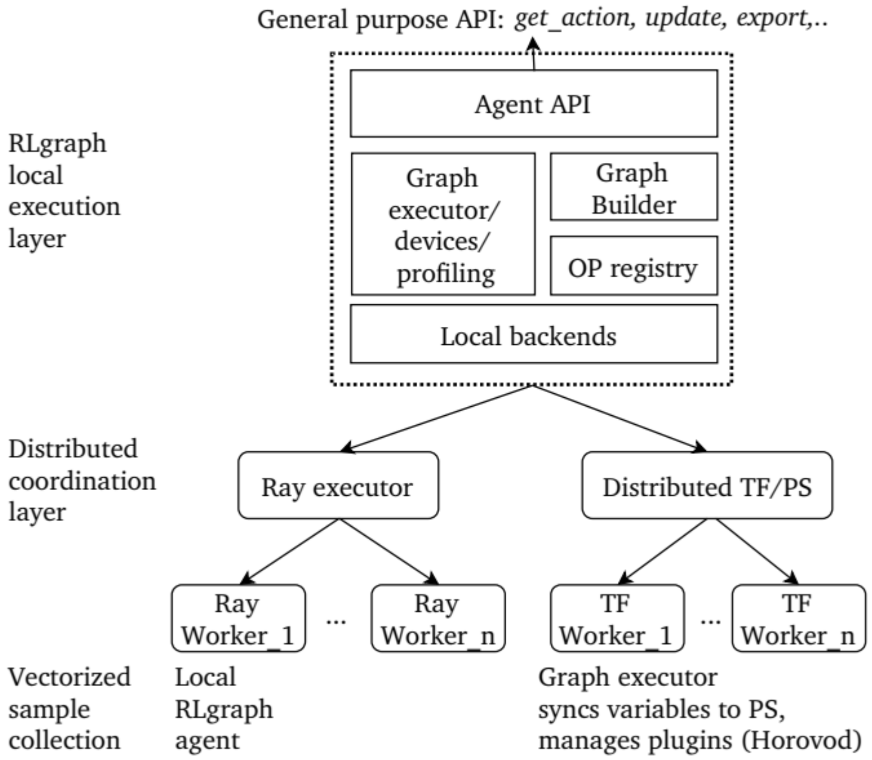
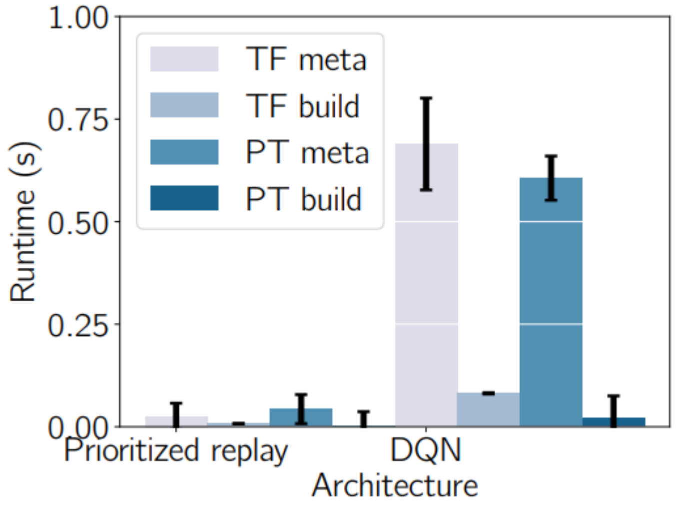



*Figure 3.* RLgraph execution stack.

# Benchmarks

1s of build overhead.

Performance wasn't sacrificed in maintaining abstraction.




(a) Build overheads.

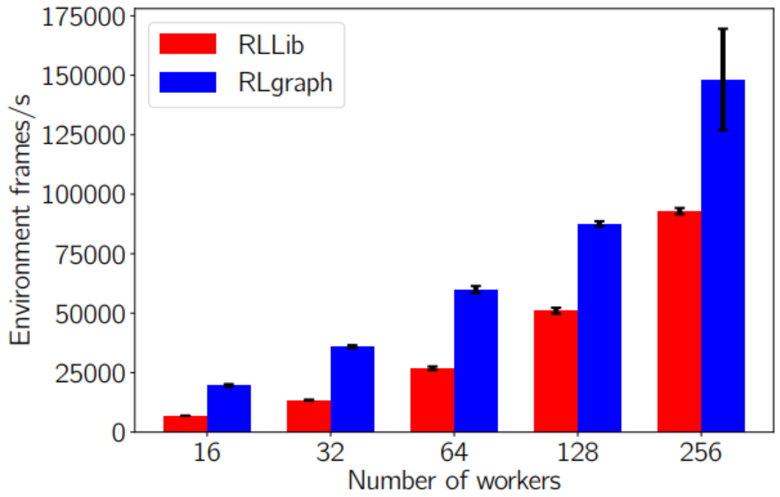



*Figure 5.* Distributed sample throughput on *Pong*.

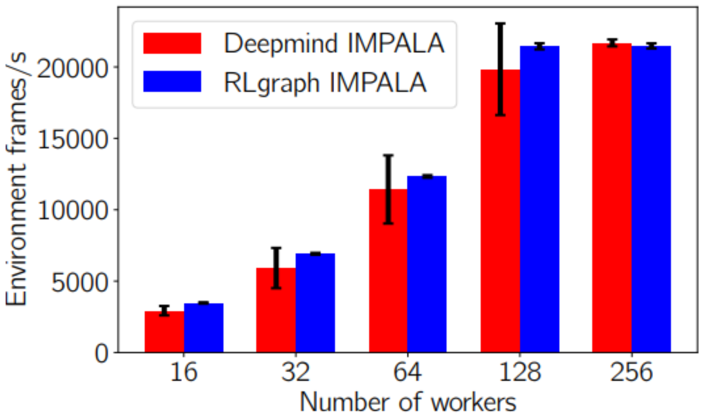



*Figure 8.* IMPALA throughput comparison on *seekavoid_arena_01*

# Critique

- Misleading to advertise a 180% speed-up on the front page compared to RLlib that is an optimization that could be incorporated into RLlib too.

- No comparison to other frameworks when driving multiple GPUs.

- No data to back up the claim that “overhead [in excessive call graph traversal while using PyTorch] becomes negligible as batch size increases and runtime is dominated by the network forward passes”.

- Nor is there any data presented on the “edge-contractions” which is the proposed mitigation of said issue.

*fin*