ELE 548: Project Proposal

Calvin Higgins

November 19, 2025

1 Completed Tasks

1.1 Learn Reinforcement Learning

As I have no prior reinforcement learning experience, I completed a reinforcement learning tutorial from the `pytorch` documentation. I trained a deep Q-network (DQN) to balance a pole on a cart by moving left or right.

1.2 Install CompilerGym

I installed CompilerGym [2], the standard framework for reinforcement learning in compilers. This was surprisingly difficult: CompilerGym is not actively maintained, many of its requirements are deprecated, and compilers are generally challenging to install correctly. I got CompilerGym working with `docker` using GPU passthrough and the `uv` package manager.

1.3 Implement DQN

To keep things simple and fast, I implemented a DQN as my initial model and targeted IR instruction count reduction.

1.4 Run Initial Experiments

I verified my implementation by optimizing IR instruction counts on several benchmark suites. To keep things simple and fast, I did not use a held-out test set. After (painful amounts of) tuning, I was able to successfully overfit a single benchmark (quicksort), the CHStone suite [6] (12 benchmarks) and the cBench suite (23 benchmarks) [4], outperforming `-Oz` on average. I was not able to outperform `-Oz` on the MiBench suite [5] (40 benchmarks). Initially, I thought that my model was struggling to learn as the number of programs increased, however, it turns out that it's just very hard to beat `-Oz` on MiBench [3]. After realizing this, I tried the Tensorflow suite [1] (1985 benchmarks), and beat `-Oz` on average. Notably, my training run crashed because the size of an optimized program became zero, indicating either (1) a compiler bug or (2) a Tensorflow bug. Unfortunately, I did not record enough data to determine which actually happened.

2 Remaining Tasks

2.1 Monitoring Infrastructure

2.2 Feature Representation

2.3 Evaluation

2.4 EfficientZero

References

[1] Martín Abadi et al. "TensorFlow: a system for large-scale machine learning". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI).* 2016. ISBN: 9781931971331.

[2] Chris Cummins et al. "CompilerGym: robust, performant compiler optimization environments for AI research". In: *International Symposium on Code Generation and Optimization (CGO).* 2022. DOI: `10.1109/CGO53902.2022.9741258`.

[3] Chaoyi Deng et al. "CompilerDream: Learning a Compiler World Model for General Code Optimization". In: *Conference on Knowledge Discovery and Data Mining (KDD).* 2025. DOI: `10.1145/3711896.3736887`.

[4] Grigori Fursin. "Collective Tuning Initiative". In: *GCC Developers Summit.* 2009.

[5] M.R. Guthaus et al. "MiBench: A free, commercially representative embedded benchmark suite". In: *IEEE International Workshop on Workload Characterization (WWC).* 2001. DOI: `10.1109/WWC.2001.990739`.

[6] Yuko Hara et al. "CHStone: A benchmark program suite for practical C-based high-level synthesis". In: *International Symposium on Circuits and Systems (ISCAS).* 2008. DOI: `10.1109/ISCAS.2008.4541637`.