ELE 548: Project Proposal

Calvin Higgins

November 20, 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 [3], 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

I tried to keep my initial implementation simple and fast. I implemented an epsilon-greedy DQN and targeted IR instruction count reduction. I used Autophase features [7] extracted from the IR code as the program representation. Since Autophase features are integer values with high variance, I standardized them with Welford's online algorithm.

1.4 Run Initial Experiments

I verified my implementation by optimizing IR instruction counts on several benchmark suites. Again, to keep the initial implementation 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 [8] (12 benchmarks) and the cBench suite (23 benchmarks) [5], outperforming `-Oz` on average. I was not able to outperform `-Oz` on the MiBench suite [6] (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 [4]. 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

I implemented some infrastructure for monitoring relevant metrics such as gradient norms and losses. However, before moving on, I want to add monitoring for the reward, action, Q-value, and IR instruction count distributions to inform later design decisions. I estimate this will take around one day of work.

2.2 Feature Representation

I believe the feature representation can be greatly improved. I have several ideas I want to try including normalizing inputs and rewards to the baseline state, pretrain an autoencoder on Autophase features, and try an Transformer encoder with inst2vec embeddings [2]. I estimate this will take around one week of work.

2.3 Evaluation

Currently, I am not evaluating on a held-out test set. This is bad for comparison but fine for experimentation since phase-ordering is very hard to overfit. I need to add evaluation on a held-out test set. I estimate this will take around one day of work.

2.4 EfficientZero

I need to replace the DQN with EfficientZero [9]. I estimate this will take around one week of work.

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] Tal Ben-Nun, Alice Shoshana Jakobovits, and Torsten Hoefler. "Neural code comprehension: a learnable representation of code semantics". In: *International Conference on Neural Information Processing Systems (NeurIPS)*. 2018.

[3] 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.

[4] 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.

[5] Grigori Fursin. “Collective Tuning Initiative”. In: *GCC Developers Summit.* 2009.

[6] 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.

[7] Ameer Haj-Ali et al. “AutoPhase: Juggling HLS Phase Orderings in Random Forests with Deep Reinforcement Learning”. In: *Machine Learning and Systems (MLSys).* 2020.

[8] 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.

[9] Weirui Ye et al. “Mastering Atari Games with Limited Data”. In: *International Conference on Neural Information Processing Systems (NeurIPS).* 2021.