Applying Deep RL Towards Compiler Optimization Problems

CompilerGym

Alfred Timperley, Brennan Richards, Ray Turrisi

Outline

Objective & Motivation

Objective

Objective:

- Use Deep RL to outperform conventional methods in the application of compiler optimization functions.
- Explore the efficacy of existing tools applied towards these problems, and identify how we could move forward with different models and strategies.

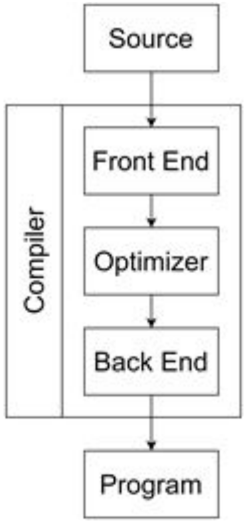

Figure 1: Abstract Representation of an Optimizing Compiler.

Motivation

- There is an increasing interest in improving analytical methods and fundamental heuristics for optimizing compiler performance
- Cloud computing centers store large amounts of data and programs
- Costs include energy, cooling, staffing (maintenance/repair)
- A small increase in compiler efficiency will boost the capacity of cloud compute centers and save money

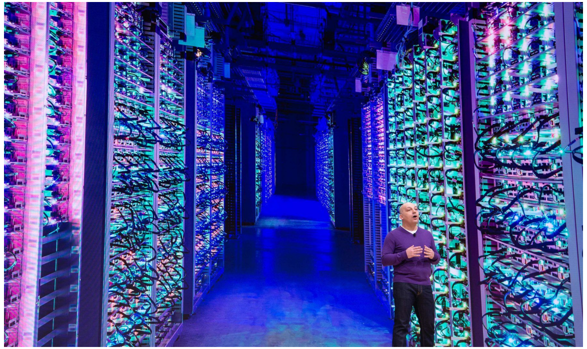

Figure 2: Google cloud computing center, with Majd Bakar (a Google VP) (Source).

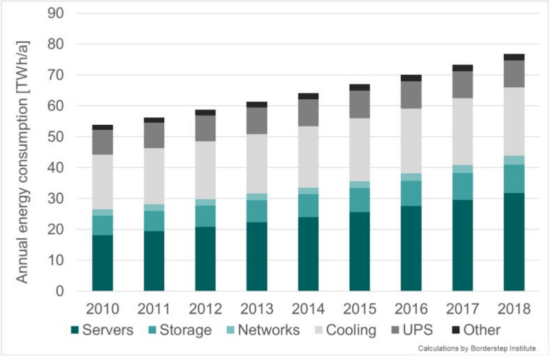

Figure 3: Increasing energy usage of cloud computing technologies (Source).

Background

Why Applying Deep RL Towards Compiler Optimization Problems?

- Reinforcement learning is a machine learning technique which trains generalized models to navigate a complex solution space

- Goal: Design an agent which can interpret a programs state and features, and assess how it can be optimized best towards some objective

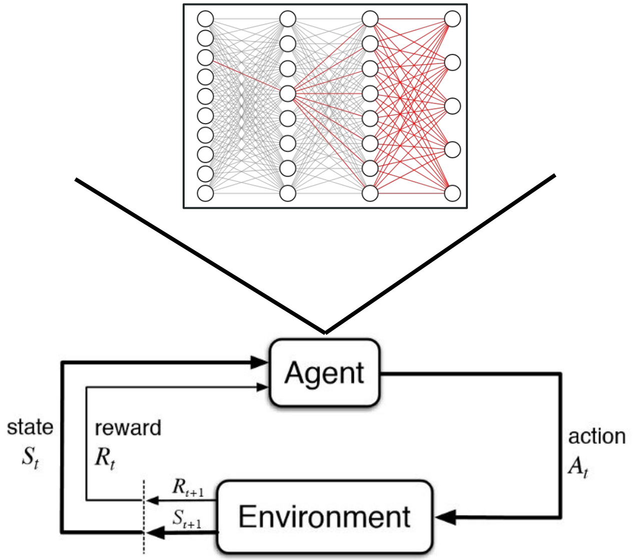

Figure 4: Reinforcement Learning fundamental Agent-Environment interface concept.

Phase Ordering

"Which number of optimizations should be applied to the program, and in which order, to achieve the greatest benefit?"

- Optimizations can be applied…
  - In any order
  - Multiple times in the same sequence
- Phase ordering has an infinitely large space of possible action sequences
- The **-Oz** flag provides a pre-defined sequence of optimizations meant to aggressively reduce the **size** of the compiled code, which was derived analytically to determine the average-best order

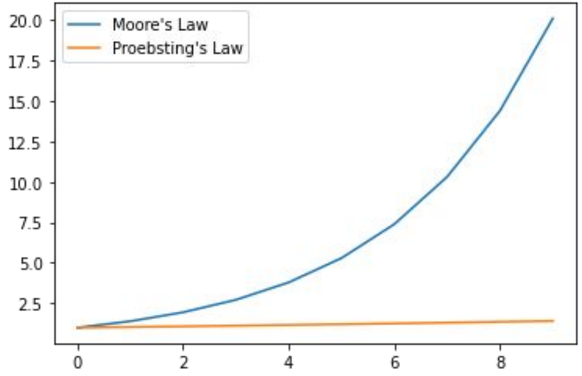

Figure 5: Moore's Law stated that computational power would continue to double every two years, while Proebsting's Law states that improvements to compiler technology double performance of typical programs every 18 years (Source).

Tools

Compiler Gym

- Custom OpenAI Gym API with many features

- Minimizes need for compiler expertise, can directly apply machine learning methods

CompilerGym

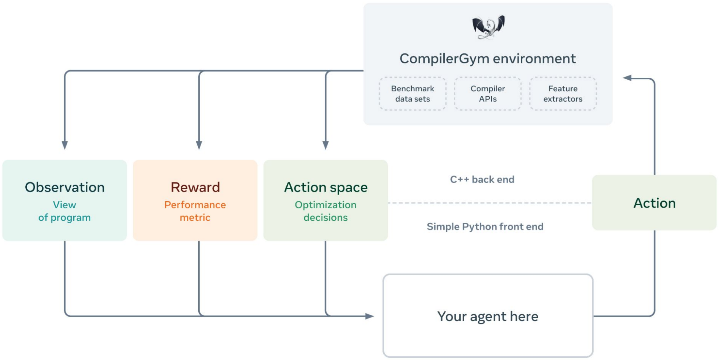

Figure 6: Compiler Gym design and usage architecture.

Optuna

- An open source hyperparameter optimization framework

- Allowed us to **parallelize hyperparameter searches**

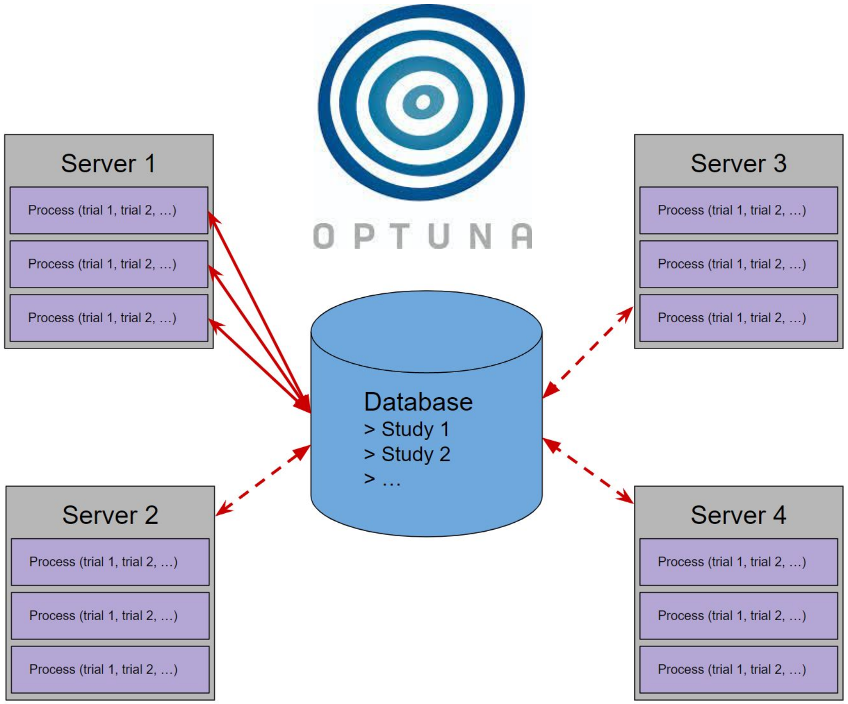

Figure 7: Optuna hyperparameter search parallelization methods.

Optuna

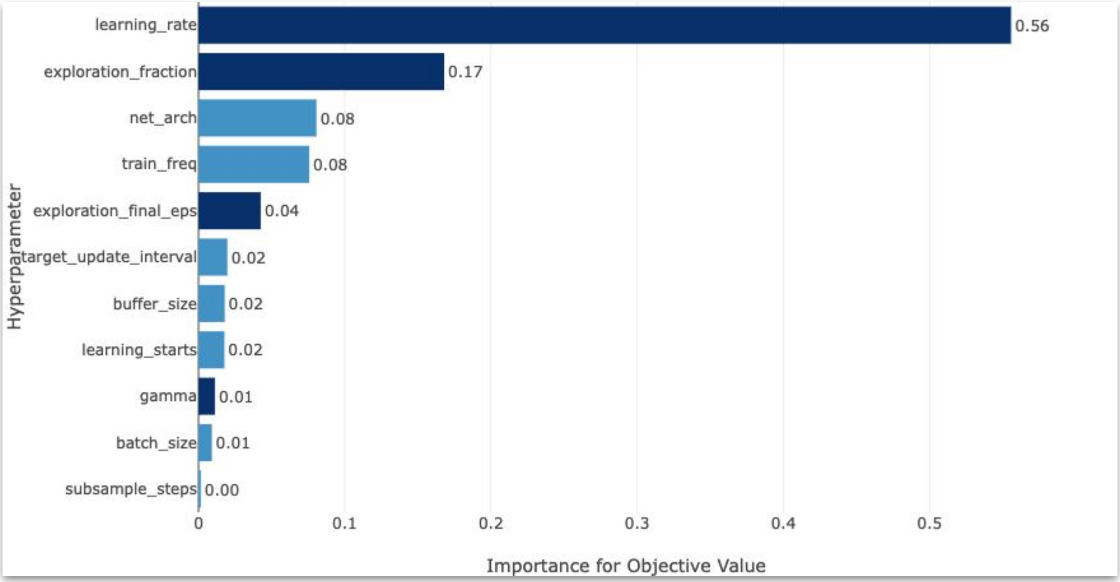

Figure 8: Hyperparameter importance plot after 6,000 trials of DQN.

Stable-baselines3

- Set of reliable deep RL algorithm implementations
- Based on OpenAI’s “Baselines” library, but with more features and more reliable
- Algorithms used:
    - DQN: Deep Q-Network Learning
    - A2C: Actor Critic Learning
    - PPO: Proximal Policy Optimization

Figure 9: Stable-baselines’ logo.

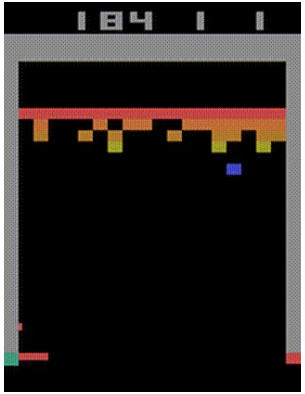

Figure 10: Trained A2C agent on Breakout.

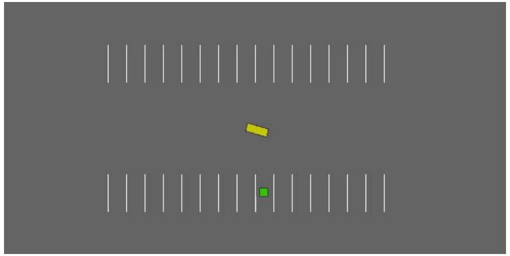

Figure 11: Hindsight Experience Replay model trained on an OpenAI Gym highway-parking-v0 environment. Less than 20000 episodes to learn how to park better than Ray.

Our Work & Results

Our Work

- Data & Environment Exploration
    - Highest reward actions
    - Code similarity vs. Model performance
- Model building and process improvement
    - Developing studies and running trials
    - Training models and wrapping their use
- Results
    - Not too bad!

Data Exploration (1)

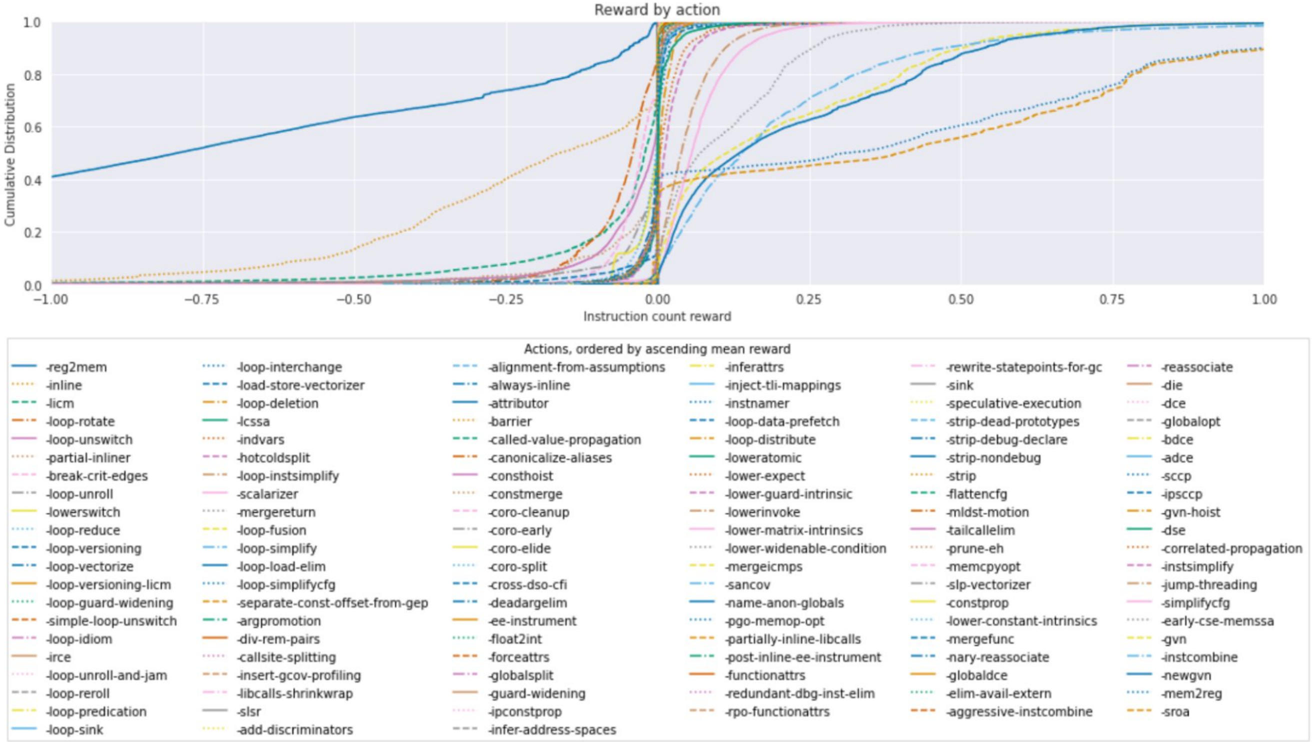

Figure 12: Optimizations with the greatest reward for code-size reduction (Source).

Data Exploration (2)

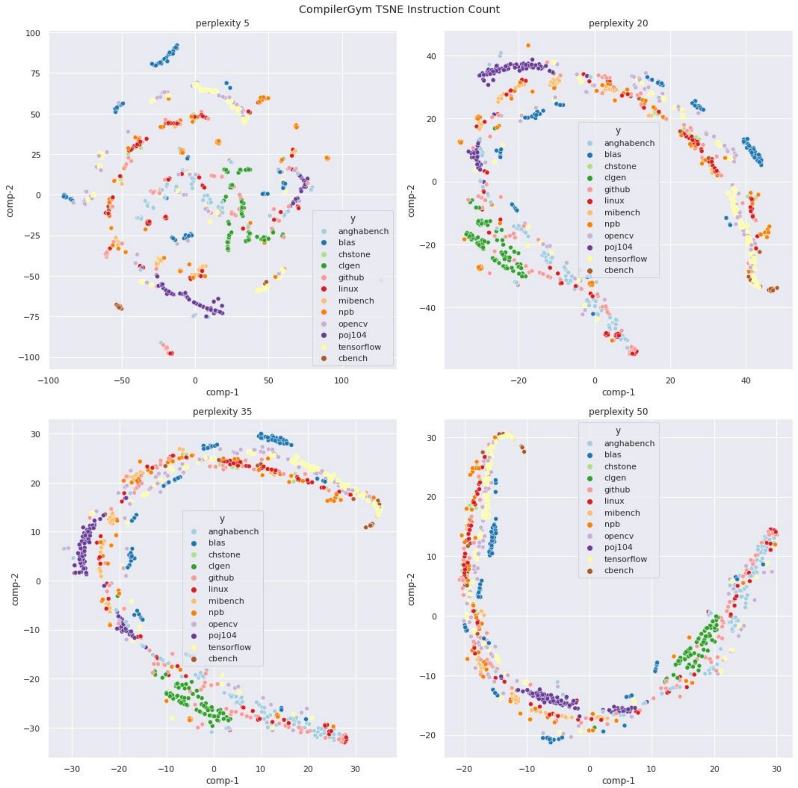

Figure 13: T-Distributed Stochastic Neighbor Embedded plot, investigating code similarity of available benchmarks.

Methods (1)

- Optuna hyperparameter optimization
  - Ran ~18 studies (> 25,000 trials), tuning:
    - Algo.-specific parameters
    - NN hyperparameters
    - Environment parameters

- Model evaluation: evaluated models on the standard test dataset, to measure results w/rt FB Research Leaderboards

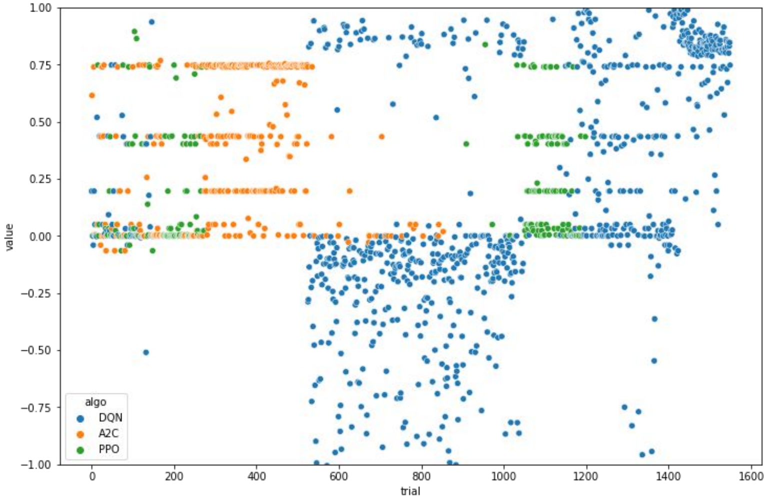

Figure 14: Optuna hyperparameter optimization for 1 study and up to 1600 trials. Here, we tested three algorithms and various hyper parameters. Towards the end of the study, we were achieving consistent results and clusters of similar hyper parameters and results.

Methods (2)

- Trained on 3 high performance computers and a supercomputer (Bridges-2)
- Studies would range from 1 day to 1 week
- Datasets:
  - Isolated cbench
  - Removed generators
  - Trained on remaining benchmarks
  - Tested on cbench (similar to FB Research's leaderboard)
- Randomly sampled 5000 benchmarks, and changed the benchmark after each episode to improve model generality

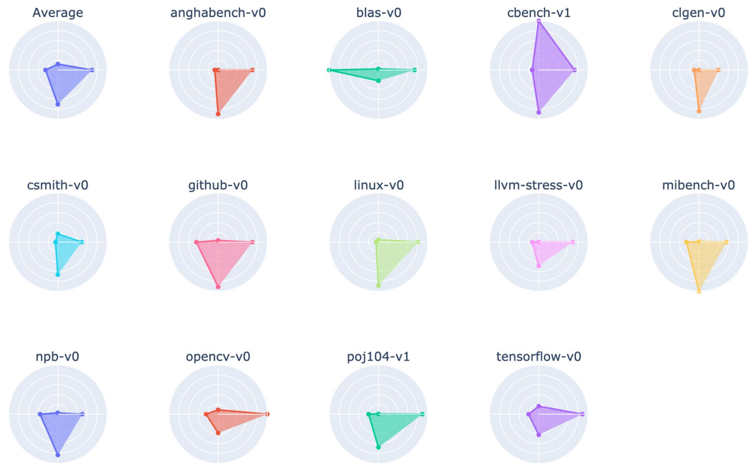

Figure 15: Compiler Gym's available datasets, benchmarks, and generators. Displaying code similarity across different datasets.

Hyperparameter Correlation (1)

- Earlier but pivotal study, where we went from 0.51x to 0.846x reward by allowing Optuna to consider different activation functions and NN depths
- In general, model hyperparameters were weakly correlated with the reward

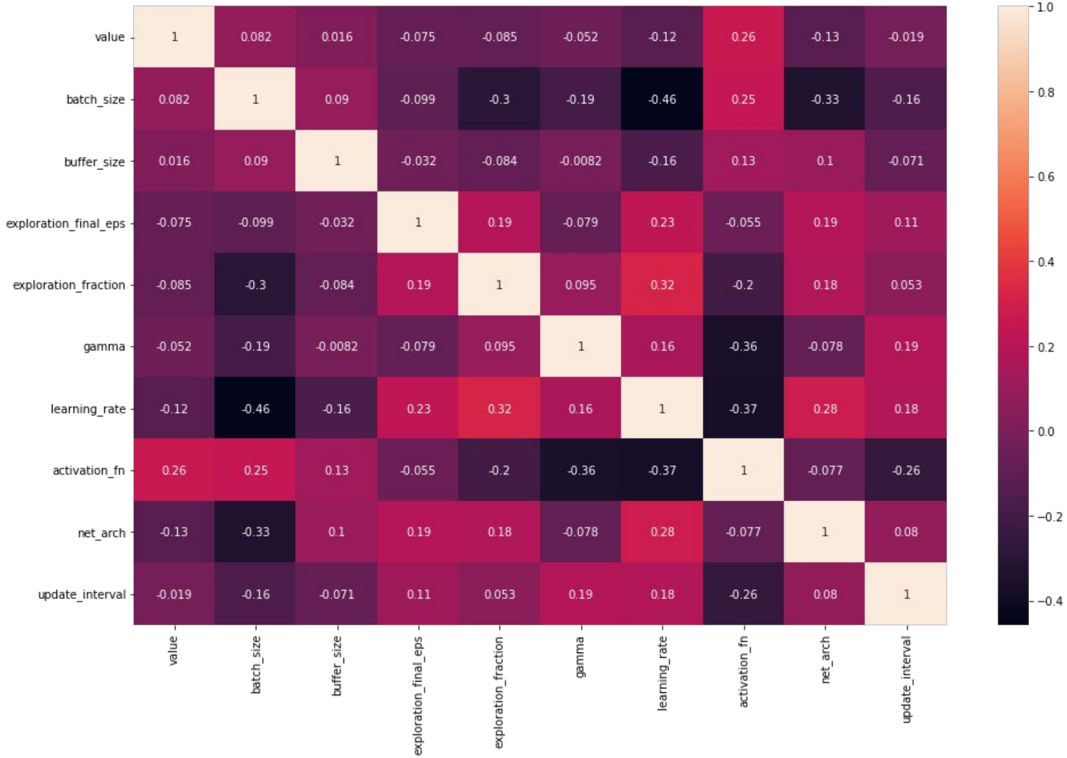

Figure 16: Pearson correlation matrix between various DQN hyper parameters, from one of our earlier but pivotal studies.

Hyperparameter Correlation (2)

- Correlation matrix from most successful study
- Tightened the range of possible hyperparameters learned from previous trials
- Changed the observation space

>> Data representation yields the greatest impact on model performance

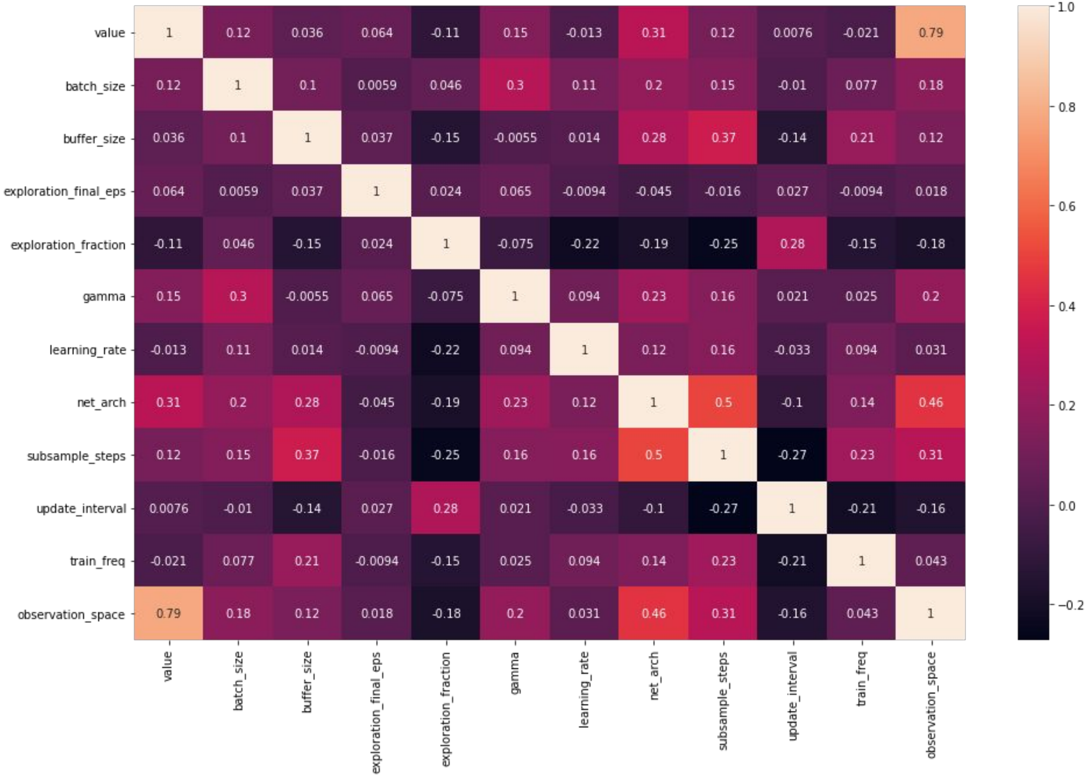

Figure 17: Pearson correlation matrix from our most successful study, revealing that the observation space had a greatest correlation with the results.

Best Model

- DQN
    - 1.005x
    - Trained on 45k episodes in 1 hours 35 minutes on the Bridges-2 supercomputer
    - The top 100 models (out of 1600 for this particular study) were all DQN, ranging from 0.85x to 1.005x.
    - Had similar hyperparameters

	Number	State	Value ↓	Duration(ms)	Params
⌄	1413	Complete	1.0050818692572647	5135354	action_subset: leaderboard_actions, algo_type: DQN, dqn_batch_size: 256, dqn_buffer_size: 100000, dqn_exploration_final_eps: 0.1796559659051645, dqn_exploration_fraction: 0.48572600600279464, dqn_gamma: 0.995, dqn_learning_rate: 0.0006760923694541781, dqn_learning_starts: 0, dqn_net_arch: medium, dqn_subsample_steps: 2, dqn_target_update_interval: 15000, dqn_train_freq: 256, n_parallel_envs: 4, observation_space: InstCountNorm

Figure 18: Zoomed-in best performing trial.

	Number	State	Value ↓	Duration(ms)	Params
⌄	1413	Complete	1.0050818692572647	5135354	action_subset: leaderboard_actions, algo_type: DQN, dqn_batch_size: 256, dqn_buffer_size: 100000, dqn_exploration_final_eps: 0.1796559659051645, dqn_exploration_fraction: 0.48572600600279464, dqn_gamma: 0.995, dqn_learning_rate: 0.0006760923694541781, dqn_learning_starts: 0, dqn_net_arch: medium, dqn_subsample_steps: 2, dqn_target_update_interval: 15000, dqn_train_freq: 256, n_parallel_envs: 4, observation_space: InstCountNorm
⌄	1437	Complete	1.0003477152931737	9189700	action_subset: leaderboard_actions, algo_type: DQN, dqn_batch_size: 256, dqn_buffer_size: 100000, dqn_exploration_final_eps: 0.18872358265244693, dqn_exploration_fraction: 0.09013637856870513, dqn_gamma: 0.98, dqn_learning_rate: 0.0016478376683243263, dqn_learning_starts: 0, dqn_net_arch: medium, dqn_subsample_steps: 2, dqn_target_update_interval: 15000, dqn_train_freq: 16, n_parallel_envs: 2, observation_space: InstCountNorm
⌄	1199	Complete	0.9907483879040229	3683035	action_subset: cg_action_reference, algo_type: DQN, dqn_batch_size: 512, dqn_buffer_size: 50000, dqn_exploration_final_eps: 0.18604810363477464, dqn_exploration_fraction: 0.4398226450559795, dqn_gamma: 0.999, dqn_learning_rate: 0.00016019213292213234, dqn_learning_starts: 1000, dqn_net_arch: tiny, dqn_subsample_steps: 8, dqn_target_update_interval: 20000, dqn_train_freq: 1, n_parallel_envs: 4, observation_space: InstCountNorm
⌄	1429	Complete	0.9899110572179779	9250518	action_subset: leaderboard_actions, algo_type: DQN, dqn_batch_size: 256, dqn_buffer_size: 100000, dqn_exploration_final_eps: 0.04233307157202048, dqn_exploration_fraction: 0.4868227436152491, dqn_gamma: 0.98, dqn_learning_rate: 0.0007167262599084408, dqn_learning_starts: 0, dqn_net_arch: medium, dqn_subsample_steps: 2, dqn_target_update_interval: 15000, dqn_train_freq: 4, n_parallel_envs: 2, observation_space: InstCountNorm
⌄	1446	Complete	0.9892650857043919	8504428	action_subset: leaderboard_actions, algo_type: DQN, dqn_batch_size: 256, dqn_buffer_size: 100000, dqn_exploration_final_eps: 0.1894487797522927, dqn_exploration_fraction: 0.4859996553604364, dqn_gamma: 0.95, dqn_learning_rate: 0.001750017326453419, dqn_learning_starts: 0, dqn_net_arch: medium, dqn_subsample_steps: 2, dqn_target_update_interval: 15000, dqn_train_freq: 16, n_parallel_envs: 2, observation_space: InstCountNorm
⌄	1431	Complete	0.9892303151515079	9366201	action_subset: leaderboard_actions, algo_type: DQN, dqn_batch_size: 256, dqn_buffer_size: 100000, dqn_exploration_final_eps: 0.18963676683293404, dqn_exploration_fraction: 0.48898172907062354, dqn_gamma: 0.98, dqn_learning_rate: 0.000688491725354111, dqn_learning_starts: 0, dqn_net_arch: medium, dqn_subsample_steps: 2, dqn_target_update_interval: 15000, dqn_train_freq: 128, n_parallel_envs: 2, observation_space: InstCountNorm
⌄	1352	Complete	0.9885739001270849	5954913	action_subset: leaderboard_actions, algo_type: DQN, dqn_batch_size: 256, dqn_buffer_size: 100000, dqn_exploration_final_eps: 0.18971755470371343, dqn_exploration_fraction: 0.24486179791025892, dqn_gamma: 0.98, dqn_learning_rate: 0.0009052313753842884, dqn_learning_starts: 0, dqn_net_arch: medium, dqn_subsample_steps: 2, dqn_target_update_interval: 15000, dqn_train_freq: 256, n_parallel_envs: 4, observation_space: InstCountNorm
⌄	1414	Complete	0.9884509048424661	4984083	action_subset: leaderboard_actions, algo_type: DQN, dqn_batch_size: 256, dqn_buffer_size: 100000, dqn_exploration_final_eps: 0.18970175341034495, dqn_exploration_fraction: 0.4870059004564221, dqn_gamma: 0.995, dqn_learning_rate: 0.0023700232291501114, dqn_learning_starts: 0, dqn_net_arch: medium, dqn_subsample_steps: 2, dqn_target_update_interval: 15000, dqn_train_freq: 256, n_parallel_envs: 4, observation_space: InstCountNorm
⌄	1202	Complete	0.9836401638982352	4422349	action_subset: cg_action_reference, algo_type: DQN, dqn_batch_size: 512, dqn_buffer_size: 1000000, dqn_exploration_final_eps: 0.010550523817219595, dqn_exploration_fraction: 0.44535997349090795, dqn_gamma: 0.9999, dqn_learning_rate: 0.00014528921695025743, dqn_learning_starts: 1000, dqn_net_arch: small, dqn_subsample_steps: 4, dqn_target_update_interval: 20000, dqn_train_freq: 1, n_parallel_envs: 4, observation_space: InstCountNorm
⌄	1203	Complete	0.9827317648887401	4736461	action_subset: cg_action_reference, algo_type: DQN, dqn_batch_size: 512, dqn_buffer_size: 50000, dqn_exploration_final_eps: 0.010056553653155276, dqn_exploration_fraction: 0.43652122627567685, dqn_gamma: 0.9, dqn_learning_rate: 0.00013989702862292775, dqn_learning_starts: 1000, dqn_net_arch: tiny, dqn_subsample_steps: 4, dqn_target_update_interval: 20000, dqn_train_freq: 1, n_parallel_envs: 4, observation_space: InstCountNorm

Figure 19: Top 10 results from best study.

Results

Figure 20: Model improvement overtime.

Best final result

1.005

How Our Results Compare

Author	Algorithm	Links	Date	Walltime (mean)	Codesize Reduction (geomean)
Facebook	Random search (t=10800)	write-up, results	2021-03	10,512.356s	**1.062×**
Facebook	Random search (t=3600)	write-up, results	2021-03	3,630.821s	1.061×
Facebook	Greedy search	write-up, results	2021-03	169.237s	1.055×
Facebook	Random search (t=60)	write-up, results	2021-03	91.215s	1.045×
Facebook	e-Greedy search (e=0.1)	write-up, results	2021-03	152.579s	1.041×
Jiadong Guo	Tabular Q (N=5000, H=10)	write-up, results	2021-04	2534.305	1.036×
Facebook	Random search (t=10)	write-up, results	2021-03	**42.939s**	1.031×
Patrick Hesse	DQN (N=4000, H=10)	write-up, results	2021-06	91.018s	1.029×
Jiadong Guo	Tabular Q (N=2000, H=5)	write-up, results	2021-04	694.105	0.988×

We are here*

Figure 21: Leaderboard from Facebook's Compiler Gym website.

Conclusions and Future Work

- Data representation
  - Conventional models are likely to be able to perform these tasks well if data is represented more meaningfully
- Models
  - There is a need to develop more custom models and training methods which is unavailable in Stable-baselines
  - Custom models would allow greater flexibility in how we use the environment's observations
- Policy Use
  - More sophisticated applications of the models could have been used if time permitted. i.e. tree search methods using a given model to make informed guesses

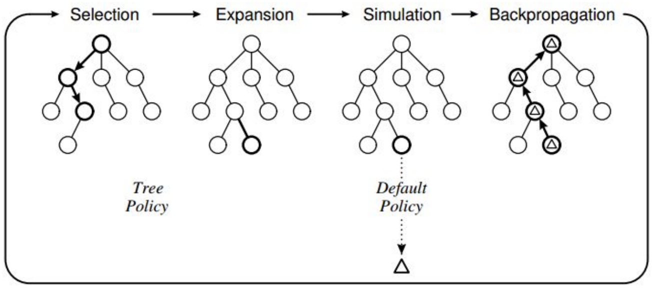

Figure 22: Tree search methods using reinforcement learning (Source).

References

- Compiler Gym: https://compilergym.com/
- Stable-baselines: https://stable-baselines3.readthedocs.io/en/master/
- Optuna: https://optuna.readthedocs.io/en/stable/tutorial/index.html
- References:

  - A. H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano, “A survey on compiler autotuning using machine learning” ACM Computing Surveys, vol. 51, no. 5, 2018.
  - C. Cummins,Deep Learning for Compilers. PhD dissertation, University of Edinburgh,2020.
  - H. Leather and C. Cummins, “Machine Learning in Compilers: Past, Present and Future,”Forum on Specification and Design Languages, vol. 2020-Septe, 2020.
  - C. Cummins, B. Wasti, J. Guo, B. Cui, J. Ansel, S. Gomez, S. Jain, J. Liu, O. Teytaud,B. Steiner, Y. Tian, and H. Leather, “CompilerGym: Robust, Performant Compiler Optimization Environments for AI Research,” pp. 1–12, 2021.

Questions