Learning Compiler Pass Orders using Coreset and Normalized Value Prediction

Youwei Liang*, Kevin Stone* , Ali Shameli, Chris Cummins, Mostafa Elhoushi, Jiadong Guo, Benoit Steiner, Xiaomeng Yang, Pengtao Xie, Hugh Leather, Yuandong Tian

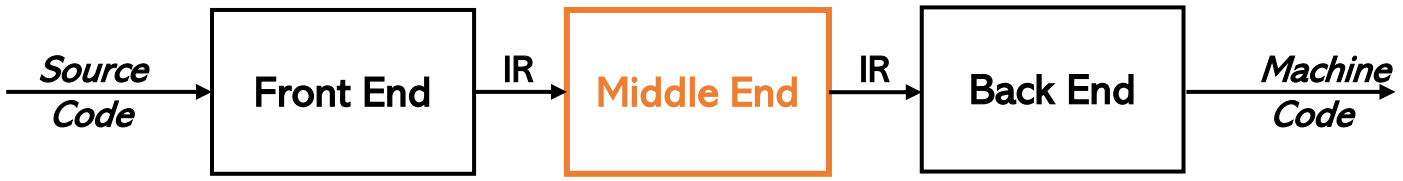
***co-first author**

Meta AI

UC San Diego

Problem Definition: Compiler Pass Ordering

- Compile a program with a sequence of optimization pass: LLVM

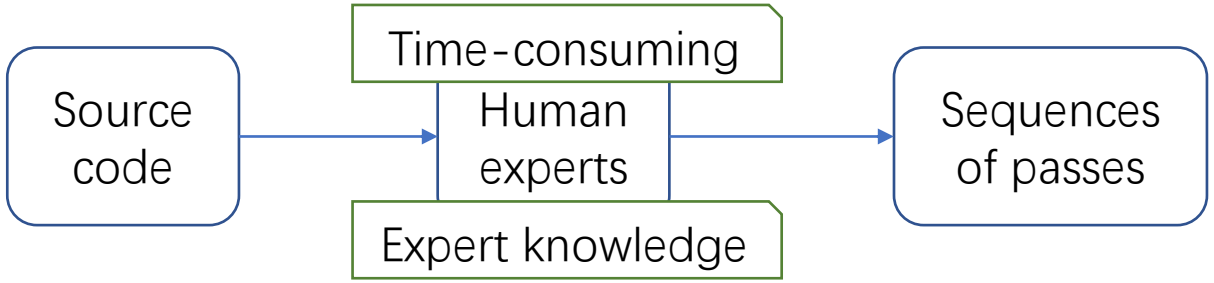


IR: intermediate representation

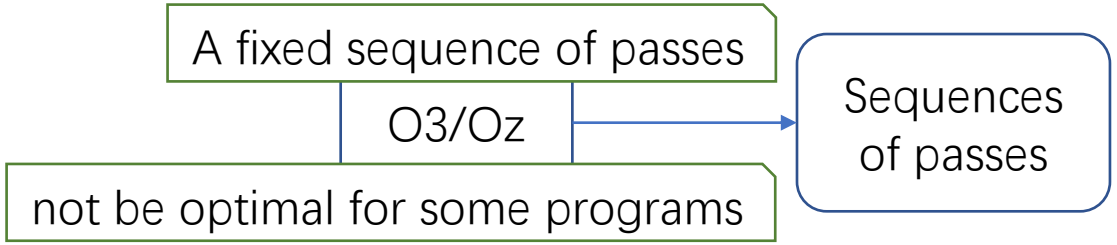
- Exemplar passes
  - Dead code elimination: e.g., remove assignments that are never used
- Order matters
  - Some passes facilitates the use of some other passes
    - -reg2mem
  - Some passes clean up the code after using certain passes
    - -dce
- In our paper, we target at minimizing code size

Challenges in Compiler Pass Ordering

- Manual optimization by humans



- Expert passes: -O3, -Oz



- Prior machine learning methods
  - Require an excessive budget at compile time (need to enumerate many passes)
    - Need to try many passess
  - Fail to generalize to unseen programs
    - Train/test on the same benchmark programs
    - Very small test set

Key Ideas of Our Approach

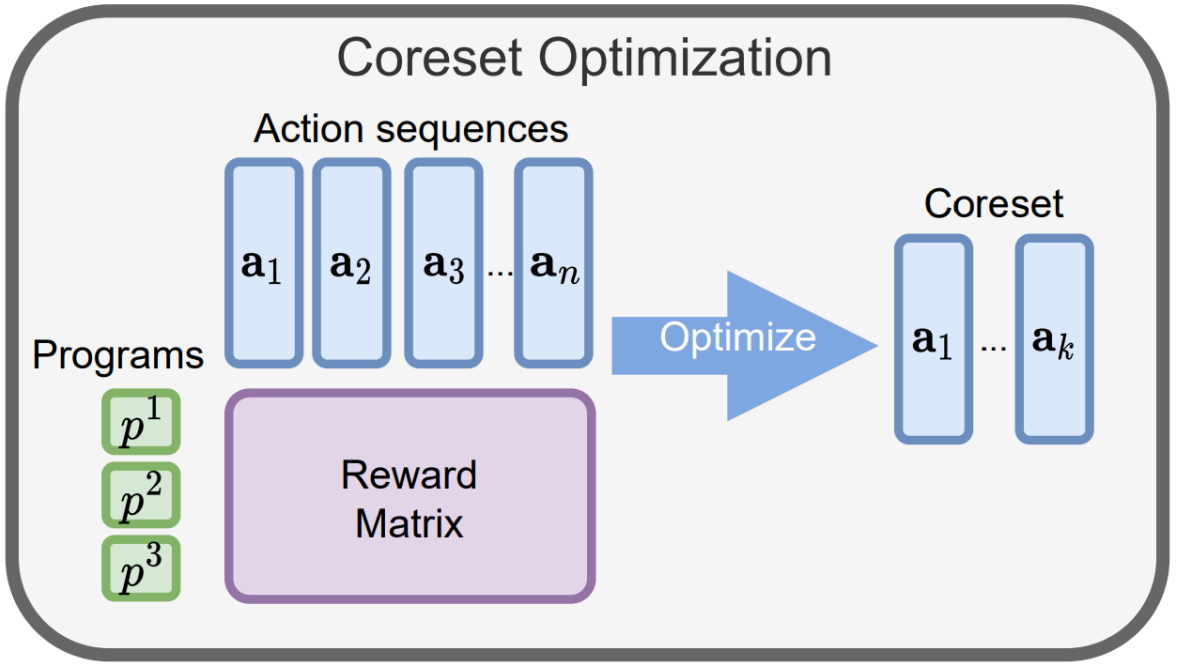
- Key ideas
  - We don’t search good passes sequentially
  - Directly find a universal core set of pass sequences (termed coreset)
  - Make decision on top of this different action space to avoid the challenge of sparse reward
- The coreset
  - Optimizes most programs
  - Requires only an upfront search cost
- The policy
  - Learns to predict the *relative* performance of coreset sequences
  - Apply the optimal coreset sequence to compile new programs at inference

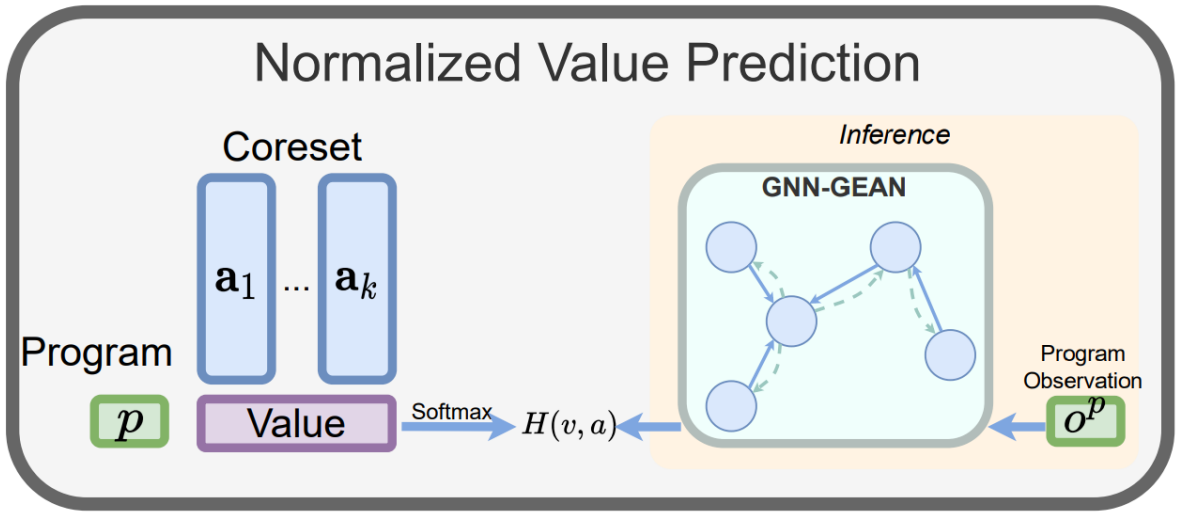
Environment Setup

- Perform pass optimization with CompilerGym
- CompilerGym - https://compilergym.com/
  - It provides a reinforcement learning (RL) environment, using similar APIs to OpenAI Gym
  - Call the APIs to obtain features/observations of a program
  - At each step, choose one of the 124 passes to apply to a program. Then environment provides the new size after applying the pass.
- Problem setup
  - Given a budget in terms of the number of passes, say 45, maximize the **size reduction** compared to the initial program
  - Allow to cache the intermediate programs at each step, so the maximum size reduction can be obtained in middle

Our Method: An Overview

- Coreset Optimization
  - Finding n candidate pass sequences
  - Finding the coreset S with a greedy algorithm
- Normalized Value Prediction
  - Given program feature, train a GNN model to output the probabilities of selecting a sequence in the coreset





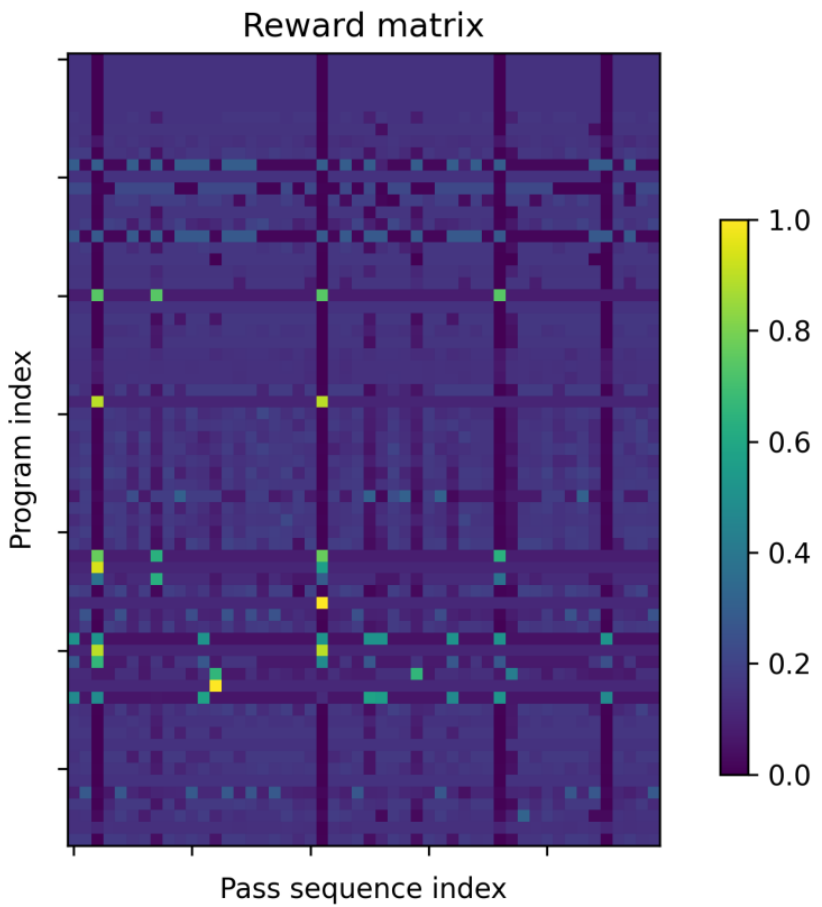
Coreset Optimization

- For each program, perform random search to find candidate pass sequences
- Construct a reward matrix $R \in \mathbb{R}^{N \times M}$: size reduction of $N = 17500$ program against $M = 17500$ candidate pass sequences
- Objective:

$$\max_{|S| \leq K} J(S) = \sum_{i=1}^{N} \max_{j \in S} r_{ij}$$

- $K = 50$: cardinality of coreset
- Initialize $S = \emptyset$
- At each step, add a new pass sequence index into S

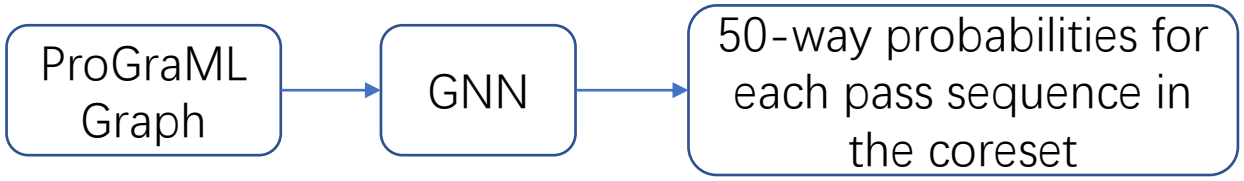
$$j_t := \arg\max_{j \notin S_{t-1}} J(S_{t-1} \cup \{j\})$$



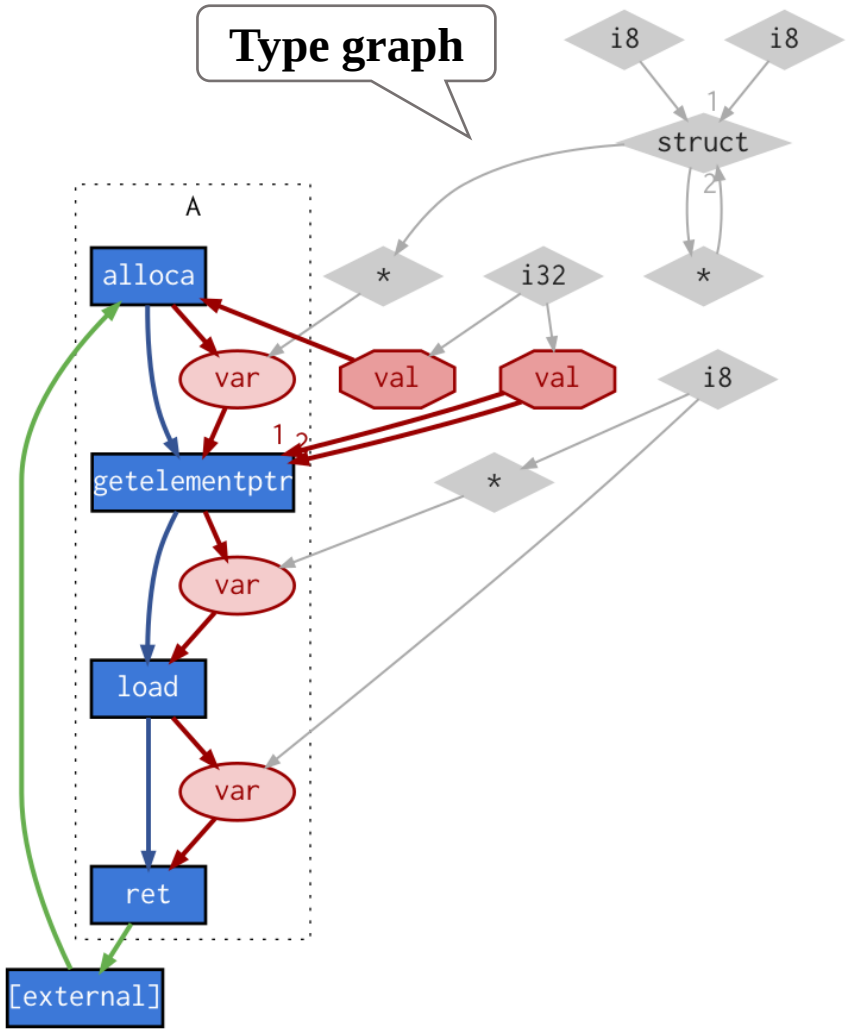
Key finding: a lot of programs can be optimized by a small number of pass sequences

Program encoding

- Input feature: ProGraML graph
- Encoder: graph neural network (GNN)
- The model selects a few **pass sequences** from the coreset to try out in the program
- This is DIFFERENT from the common practice of selecting a single pass at a time



- We also tried another input feature: Autophase vector
  - Consist of the number of certain instructions, etc
  - Use MLP for encoding



Normalized Value Prediction

- Normalize the rows of the reward matrix on the *coreset*

$$\boldsymbol{v}^p = \mathrm{Softmax}(\boldsymbol{r}^p / T)$$

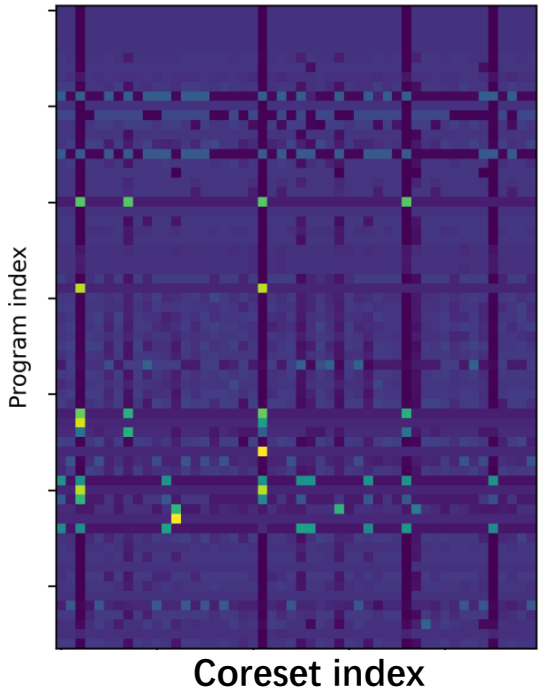
(p denotes a program)

- T is a temperature
- Loss: cross entropy between model output a^p and the normalize values v^p

$$\mathcal{L}(\boldsymbol{a}^p, \boldsymbol{v}^p) = -\sum_{j=1}^{K} v_j^p \log a_j^p$$

Key finding: a program can be optimized by several pass sequences (resulting in same size reduction)

A "slim" reward matrix



Experiments

- Datasets: a large number of diverse programs
- Allow maximum of 45 compilation passes
- Evaluation: how much the algorithm is better than Oz in terms of *code size* (IR instruction count)
- I_p^{Oz}: resulting IR instruction count of program p after applying -Oz
- $I_p^{\pi_\theta}$: resulting IR instruction count of program p using algorithm π_θ
- Mean Improvement Over Oz

$$\bar{I}^{Oz} = \text{MeanOverOz} := \frac{1}{|\mathcal{P}|} \sum_p \frac{I_p^{Oz} - I_p^{\pi_\theta}}{I_p^{Oz}}$$

- Geometric Mean Improvement Over Oz

$$\bar{I}_G^{Oz} = \text{GMeanOverOz} := \left(\prod_p \frac{I_p^{Oz}}{I_p^{\pi_\theta}} \right)^{\frac{1}{|\mathcal{P}|}}$$

Type	Dataset	Train	Val	Test
Uncurated	anghabench-v1	707,000	1,000	2,000
	blas-v0	133	28	29
	github-v0	7,000	1,000	1,000
	linux-v0	4,906	1,000	1,000
	opencv-v0	149	32	32
	poj104-v1	7,000	1,000	1,000
	tensorflow-v0	415	89	90
	clgen-v0	697	149	150
	csmith-v0	222	48	48
	llvm-stress-v0	697	149	150
Curated	cbench-v1	0	0	11
	chstone-v0	0	0	12
	mibench-v1	0	0	40
	npb-v0	0	0	121
Total	-	728,219	4,495	4,683

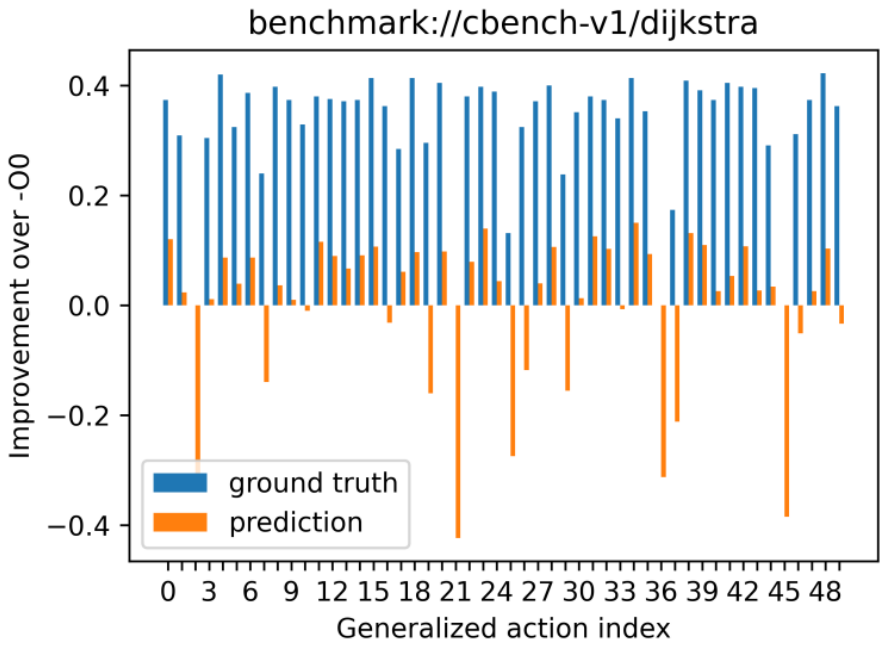
Experiments

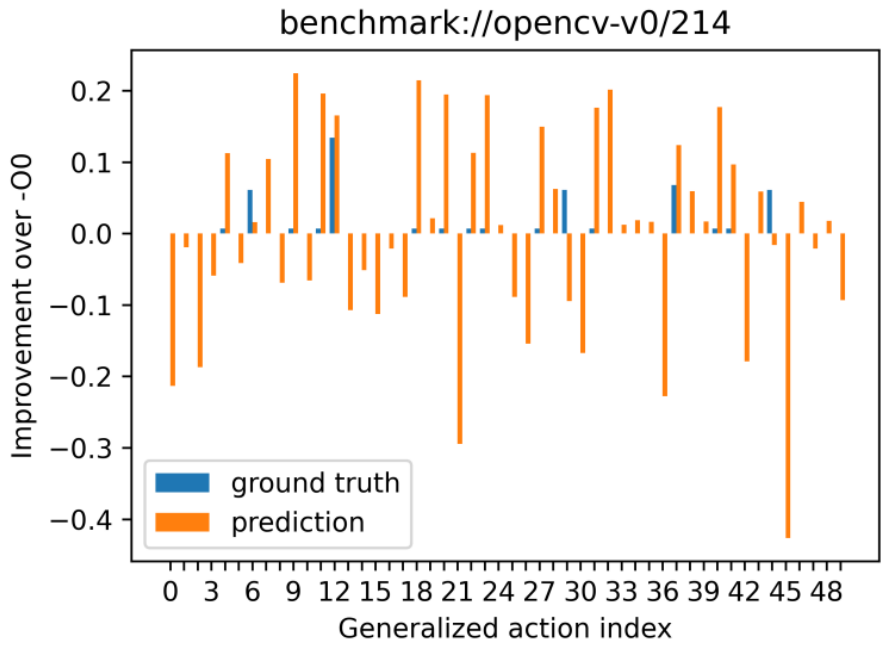
- Baselines
  - Oracle: the best in the coreset
  - Top-45: only allow 45 steps in using the coreset
  - RL-PPO: proximal policy optimization
  - Q-value-rank: directly learn the values of the pass sequences in the coreset
  - BC: behavior cloning; classification over the coreset
  - NVP: normalized value prediction
- For Top-45, Q-value-rank, BC, and NVP
  - The budget of 45 steps allows us to use roughly 4 pass *sequences*; roll out them one by one
  - Truncate the last sequence applied (not to exceed the budget)
- GNN: GCN, GGC, GIN, GAT, GEAN

Method	#passes	$\bar{I}^{Oz}(\%)$	$\bar{I}_G^{Oz}$
Compiler (`-Oz`)	97	0	1.000
`Autophase-PPO`	45	$-16.3_{\pm 9.8}$	$0.960_{\pm 0.036}$
`GCN-PPO`	45	$-10.3_{\pm 1.0}$	$0.998_{\pm 0.003}$
`GGC-PPO`	45	$-12.3_{\pm 0.1}$	$0.988_{\pm 0.001}$
`GIN-PPO`	45	$-15.1_{\pm 5.9}$	$0.972_{\pm 0.029}$
`GAT-PPO`	45	$-65.7_{\pm 40.1}$	$0.806_{\pm 0.132}$
`GEAN-PPO`	45	$-12.0_{\pm 0.6}$	$0.997_{\pm 0.002}$
`Autophase-Q`	45	$-3.9_{\pm 0.2}$	$1.006_{\pm 0.002}$
`GEAN-Q`	45	$0.7_{\pm 1.3}$	$1.016_{\pm 0.012}$
`Autophase-BC`	45	$2.9_{\pm 0.1}$	$1.045_{\pm 0.000}$
`GEAN-BC`	45	$2.8_{\pm 0.6}$	$1.045_{\pm 0.007}$
`Autophase-NVP`	45	$4.0_{\pm 0.4}$	$1.056_{\pm 0.005}$
`GCN-NVP`	45	$4.3_{\pm 0.1}$	$1.058_{\pm 0.001}$
`GGC-NVP`	45	$4.4_{\pm 0.2}$	$1.059_{\pm 0.002}$
`GIN-NVP`	45	$4.3_{\pm 0.3}$	$1.058_{\pm 0.003}$
`GAT-NVP`	45	$4.5_{\pm 0.2}$	$1.060_{\pm 0.001}$
`GEAN-NVP`	45	$4.5_{\pm 0.1}$	$1.059_{\pm 0.000}$
Top-45	45	-7.5	0.992
Oracle	625	5.8	1.075

Analysis

- Learned Q values





Ablations

- Temperature
- A smaller temperature works better
- Essentially, it becomes a multi-class classification for a smaller temperature