Ordering Optimization Passes with Reinforcement Learning for Instruction Count Reduction

Calvin Higgins
Department of Computer Science and Statistics
University of Rhode Island
Kingston, United States
calvin_higgins2@uri.edu

***Abstract*—Attaining optimal program performance on modern heterogeneous hardware requires specializing compiler optimizations to individual architectures. To ease the engineering burden, researchers have proposed automatically learning optimal orderings of compiler optimization passes. In this work, we minimize IR instruction count by ordering LLVM optimization passes with deep Q-learning, a standard reinforcement learning method. We attain an TODO% reduction in instruction count compared to `-O0` (% regression from `-Oz`) across six representative benchmark suites.**

***Index Terms*—TODO**

I. Introduction

The goal of a compiler is simple: find an optimal translation of high-level code into machine code. Many modern compilers rely on the LLVM compiler infrastructure which provides a unified intermediate representation (think architecture-independent assembly language) as well as associated optimization and code-generation tools [8]. These compilers typically begin with a relatively naive translation from high-level code into LLVM intermediate representation (IR), and iteratively refine it with transformations known as optimization passes. Different optimization pass orders can yield binary with drastically different sizes and runtime performance.

The phase-ordering problem involves determining the best order to apply a fixed set of optimization passes. Historically, expert compiler engineers have designed new optimization pass orders for every architecture. However, as hardware rapidly evolves and becomes increasingly heterogeneous, maintaining optimization pass orders imposes a growing engineering burden. Automation can alleviate this burden, allowing compiler experts to focus on more critical tasks [14]. Unfortunately, as LLVM exposes hundreds of distinct passes, and typical orders apply hundreds of passes, automatically finding good orders is extremely challenging. Deep reinforcement learning has successfully explored other difficult search spaces, such as video games, demonstrating promise for compiler optimization [5].

By formulating the phase-ordering problem as a Markov decision process, standard reinforcement learning techniques apply. A Markov decision process is a four tuple (S, A, T, R) where S is a set of states, A is a set of actions, $T(s, a)$ is a transition function mapping from a state s and an action a to a new state, and $R(s, a)$ is the change in state cost upon applying action a to state s. In the phase ordering problem, the set of states S is an equivalence class of programs, the action space A is the set of all optimization passes, the transition function $T(s, a)$ outputs the program s after applying optimization pass a, and the reward function $R(s, a) = C(s) - C(T(s, a))$ where $C(s)$ is a cost function such as binary size, instruction count, or execution time. Unfortunately, not only does the large action space A lead to a combinatorial explosion of possible optimization sequences, but rewards are also sparse as effective sequences are rare, and evaluating the transition and reward functions is slow as they require compiler invocations.

Accordingly, any approach to the phase-ordering problem ought to be sample efficient. CompilerDream [3], a state-of-the-art phase-ordering policy, adapted DreamerV2 [4], a general purpose reinforcement learning method to compiler optimization. However, although the Dreamer methods dominate nearly all reinforcement learning methods across a diverse set of domains, they are notably overshadowed by EfficientZero [15] in low sample regimes [5]. Motivated by its sample efficiency, we aim to apply EfficientZero to the phase-ordering problem. In this work, we conduct preliminary experiments with deep Q-learning [11], hoping to gain experience and insights before implementing EfficientZero.

II. Related Work

References

[1] Mohammed Almakki et al. “Autophase V2: Towards Function Level Phase Ordering Optimization”. In: *Machine Learning for Computer Architecture and Systems (MLArchSys)*. 2022.

[2] Tianming Cui et al. “DeCOS: Data-Efficient Reinforcement Learning for Compiler Optimization Selection Ignited by LLM”. In: *International Conference on Supercomputing (ICS)*. 2025. DOI: 10.1145/3721145.3725765.

[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] Danijar Hafner et al. “Mastering Atari with Discrete World Models”. In: *International Conference on Learning Representations (ICLR)*. 2021.

TABLE I
SUMMARY OF PRIOR WORK

Method	**Objectives**	**Benchmarks**	**Models**	**Representation**	**Rewards**	**Results**
AutoPhase [6]	Cycle count	High-Level Synthesis	PPO, A3C, Random Forest	AutoPhase features, action history	$\log \frac{C(s_{t-1})}{C(s_t)}$	1.28x speedup vs. `-O3`
CORL [10]	Runtime	LLVM test suite	Deep Q-learning	inst2vec, action history	$\log \frac{C(s_{t-1})}{C(s_t)}$	Slowdown vs. `-O3`
POSET-RL [7]	Throughput, binary size	SPEC-CPU 2006, SPEC-CPU 2017, MiBench	Deep Q-Learning	IR2Vec	Linear combination of $\frac{C(s_{t-1})-C(s_t)}{C(s_{\texttt{-O0}})}$ for throughput, binary size	1.04x throughput, 1.03x compression vs. `-Oz`
AutoPhase V2 [1]	IR instruction count	cBench, CHStone, Csmith	PPO	AutoPhase features, action history	$\frac{C(s_t)-C(s_{t-1})}{C(s_{\texttt{-O0}})-C(s_{\texttt{-Oz}})}$	1.00x compression vs. `-Oz`
GEAN [9]	IR instruction count	AnghaBench, BLAS, GitHub, Linux, OpenCV, POJ-104, TensorFlow, CLGen, Csmith, LLVM-Stress, cBench, CHStone, MiBench, NPB	Coreset, GEAN (GAT-like GNN)	ProGraML	Minimum observed cost	1.05x compression vs. `-Oz`
DeCOS [2]	Cycle count	SPEC-CPU 2017	Unspecified RL, LLM	IR2Vec, dynamic features, action history	$C(s_t) - C(s_{t-1})$, unspecified reward for final result	1.21x speedup vs. `OpenTuner`
CompilerDream [3]	IR instruction count	CodeContests, BLAS, cBench, CHStone, Linux, MiBench, NPB, OpenCV, TensorFlow	DreamerV2, guided search	AutoPhase features, action history	$\frac{C(s_t)-C(s_{t-1})}{C(s_{\texttt{-O0}})-C(s_{\texttt{-Oz}})}$	1.07x compression vs. `-Oz`
GRACE [13]	IR instruction count	BLAS, cBench, CHStone, MiBench, NPB, OpenCV, TensorFlow	Coreset, contrastive encoder, guided search	AutoPhase features	Minimum observed cost	1.10x compression vs. `-Oz`
Compiler-R1 [12]	IR instruction count	BLAS, cBench, CHStone, MiBench, NPB, OpenCV, TensorFlow	LLM, GRPO, PPO, RPP	AutoPhase features	$\frac{C(s_{\texttt{-O0}})-C(s_t)}{C(s_{\texttt{-O0}})}$, format reward	1.08x compression vs. `-Oz`

[5] Danijar Hafner et al. "Mastering diverse control tasks through world models". In: *Nature* (2025). DOI: 10.1038/s41586-025-08744-2.

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

[7] Shalini Jain et al. "POSET-RL: Phase ordering for Optimizing Size and Execution Time using Reinforcement Learning". In: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2022. DOI: 10.1109/ISPASS55109.2022.00012.

[8] C. Lattner and V. Adve. "LLVM: a compilation framework for lifelong program analysis and transformation". In: *International Symposium on Code Generation and Optimization (CGO)*. 2004. DOI: 10.1109/CGO.2004.1281665.

[9] Youwei Liang et al. "Learning compiler pass orders using coreset and normalized value prediction". In: *International Conference on Machine Learning (ICML)*. 2023.

[10] Rahim Mammadli, Ali Jannesari, and Felix Wolf. “Static Neural Compiler Optimization via Deep Reinforcement Learning”. In: *Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar)*. 2020. DOI: 10.1109/LLVMHPCHiPar51896.2020.00006.

[11] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *Nature* (2015). ISSN: 00280836.

[12] Haolin Pan et al. *Compiler-R1: Towards Agentic Compiler Auto-tuning with Reinforcement Learning*. 2025. arXiv: 2506.15701.

[13] Haolin Pan et al. *GRACE: Globally-Seeded Representation-Aware Cluster-Specific Evolution for Compiler Auto-Tuning*. 2025. arXiv: 2510.13176.

[14] Zheng Wang and Michael O’Boyle. “Machine Learning in Compiler Optimization”. In: *IEEE* (2018). DOI: 10.1109/JPROC.2018.2817118.

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