Given a program and a set of compiler optimizations (potentially with parameters), identifying the combination that leads to the fastest execution-time for a particular architecture is a long-standing problem in the field of compiler optimization. This problem is known as NP-hard [1]. Therefore, exhaustive enumeration of the search space remains unrealistic, even for small programs. As a result, programmers invest strenuous efforts seeking the optimal implementation of their code that exploits more efficiently the underlying hardware to make the code run faster.

Through experience, compiler designers have observed that certain sequences of optimizations tend to lead to faster execution-times. Unfortunately, the portability of these results is not guaranteed due to the diversity and the complexity of machine architectures. Furthermore, to reduce the complexity of this laborious workload and make it accessible to non-expert programmers and domain professionals, many techniques have been proposed to automate the discovery of the optimal optimization sequences. Such techniques are referred to in the literature as auto-tuning or auto-scheduling [2, 3].

Auto-schedulers appeal to classical search algorithms (e.g. depth first search [3], beam search [4], Monte-Carlo tree search [5]) guided by an evaluation function. Often, this evaluation function is approximated by data-driven predictive models [4, 5, 6, 7, 8, 9, 10] that offer an acceptable trade-off between accuracy and cost, given the significant limitations of analytical models. These predictive models (called cost models in the jargon) are largely trained in fully supervised, offline settings.

**Your contribution** The goal of this project is to build an auto-tuning framework for compiler flag optimization. Your contribution can be outlined in 4 stages:
1. Acquiring program features in a statistical or dynamic way. For example, this can be accomplished through the microarchitecture-independent workload characterization plugin.

2. Implementing a space exploration algorithm to determine the next-best optimization phase (a flag from a predefined list) to apply following a cost model.

3. Designing a deep learning-based cost model and training it by a no-regret algorithm in online settings to clone the behavior of the expert.

4. Providing a robust evaluation on a diversified benchmark against standard LLVM optimization levels.

**Deliverable**

*Final report:* The final report may be written in English or German. It must contain an abstract written in both English and German, this assignment and the schedule. It should include an introduction, an analysis of related work, and a complete description of the proposed approach and the conducted experiments. Three copies of the final report must be delivered to the supervisor.

*Reproducible experimental setup:* Implementations, configuration scripts and instructions to reproduce the results reported in the thesis must be delivered in electronic form. PyTorch implementations are preferred.

*Presentation:* The results of the thesis must be presented during an Advanced Computing Laboratory seminar. The presentation is capped to 30 minutes and should give an overview as well as the most important details of the work.

**Contact** If you are interested in pursuing this master thesis, please contact pueschel@ethz.ch or mleghettas@ethz.ch.

**References**


