# Section 1. Executive Summary

Compiler optimization is a significant problem in computer science for which numerous solutions have been developed and used traditionally, including predictive heuristic methods, iterative search methods and many more. However, each of them has limitations regarding architectural points, hardware points, and flexibility points. The shortcomings required a new integrated approach for the problem's solution, i.e. using Machine Learning for compiler optimization.

This research aims to investigate and explore how machine learning can find and predict the optimized compiler alternatives for executable programs and which ML models contribute to this research. We also aim to find the anticipated limitations of machine learning for finding optimized transformations and how the models impact the outcomes of the optimization problem. The research question is to what context machine learning helps find improved compiler organization and what models are most suitable for them. The following research objectives were identified for this study: Objective 1. To inspect existing literature on different approaches for compiler optimization and their shortcomings. Objective 2. To analyze the role of machine learning for compiler optimization. Objective 3. To discuss the machine learning algorithms and models and types of learning methods with their featured engineering methods to find the optimum solution for the problem. Some of the critical scholars identified from the literature review concerning using the machine for learning for compiler optimization are A. Gauci, K. Z. Adami, Zheng Wang, Michael O'Boyle, Agakov, F.; Bonilla, Thomson, J.

The research method used is both primary data and secondary data. The qualitative research method will be used for both data collection and methodology. The content analysis tool will serve as a method for data analysis.

Contents

[Section 1. Executive Summary 1](#_Toc89333398)

[Section 2. Introduction 3](#_Toc89333399)

[**1.** **Research Problem** 3](#_Toc89333400)

[**2.** **Research Aim, Research Question and Objectives** 3](#_Toc89333401)

[**1.** **Research Aim** 3](#_Toc89333402)

[**2.** **Research Question** 4](#_Toc89333403)

[**3.** **Objectives** 4](#_Toc89333404)

[**4.** **Research Audience** 4](#_Toc89333405)

[**5.** **Personal Motivation** 4](#_Toc89333406)

[Section 3. Literature Review 5](#_Toc89333407)

[**Learning and Training** 5](#_Toc89333408)

[**Deployment** 5](#_Toc89333409)

[**Criticism on thread coarsening** 6](#_Toc89333410)

[**Criticism on an iterative search** 6](#_Toc89333411)

[**Shortcomings in the literature** 6](#_Toc89333412)

[Section 4. Research Methods 7](#_Toc89333413)

[**1.** **Research Method** 7](#_Toc89333414)

[**2.** **Research Strategy** 7](#_Toc89333415)

[**3.** **Research Strategy – Data Collection** 7](#_Toc89333416)

[**4.** **Data Analysis** 8](#_Toc89333417)

[**5.** **Anticipated difficulties of the research** 8](#_Toc89333418)

[**6.** **Description of ethical issues** 8](#_Toc89333419)

[**7.** **Validity/Reliability** 9](#_Toc89333420)

[Section 5: References 9](#_Toc89333421)

# Section 2. Introduction

## **Research Problem**

Compiler optimization has been the fundamental problem in computer science since the early 1960s. Many contributors and researchers have been working to find a way to optimize compilers through various approaches. Optimizing compiler refers to minimizing or maximizing the components of the executable program, such as downsizing executing time and storage sizes and maximizing throughput. The traditional methods have been done through optimization algorithms that usually transform a user program to semantically correspondent code that runs by consuming fewer resources. One implementation included combining Active Harmony's parallel search backend with one of the CHILL compiler conversion frameworks to produce semantically equivalent code performances and automatically selects the best-running implementation (Ashouri, A.H., 2017). These methods now do not comprehend the fast-user experience and high throughput with parallel executions and require a more enhanced way to optimize the compiler. For this, Machine Learning can be used for optimizing the compilers. Machine learning is an area aimed to detect patterns, and with its dynamic aspects in optimizing compiler codes, IBM announced the very first open-source ML compiler. Since then, its methods and algorithms have been very efficient in generating alternative implementations and selecting the best one from them (Lins, F.M., Tambara, 2017). The paper is about how machine learning has become a natural fit to the optimization problem and is now developed into a research domain.

## **Research Aim, Research Question and Objectives**

## **Research Aim**

This research aims to investigate and explore how machine learning can find and predict the optimized compiler alternatives for executable programs and which ML models contribute to this research. We also aim to find the anticipated limitations of machine learning for finding optimized transformations and how the models impact the outcomes of the optimization problem.

## **Research Question**

To what context machine learning helps find improved compiler organization, and what models are most suitable for them.

## **Objectives**

The following research objectives were identified for this study:

Objective 1. To inspect existing literature on different approaches for compiler optimization and their shortcomings.

Objective 2. To analyze the role of machine learning for compiler optimization.

Objective 3. To discuss the machine learning algorithms and models and types of learning methods with their featured engineering methods to find the optimum solution for the problem.

## **Research Audience**

Numerous individuals can be impacted by this research, including the scientists, compiler developers, enterprises and the open-source community who takes optimization as their utmost priority for processing their programs in large clusters in big data centers (Anderson, J.A., 2018). This research can constructively impact the senior developers as it can help reduce development time whilst ensuring higher performances up to 20-30% faster than the conventional methods.

## **Personal Motivation**

I believe that this research about machine learning in compiler optimization will extend my knowledge and insights in the area of machine learning and deep learning with their practical use of solving complex real-world problems and, more importantly, finding solutions for NP-complete or undecidable as it requires programmer's capability to wait for processes to complete their execution. I strongly feel that my contribution can enormously help the open-source community of developers, and other fellow developers can further this work. Through this, we can reach the most optimum solution for a better cause. So it's not just my love for machine learning. It's also the output that can benefit numerous at a time.

# Section 3. Literature Review

## **Learning and Training**

Machine learning for optimization has been an exciting subject for researchers since the 1800s, and many contributions have been made for this purpose. Zheng Wang and Michael O'Boyle divide the problem into two main stages, i.e. learning and deployment where the first stage entails learning the model based on the seen data through models (Wang, Z. and O’Boyle, M., 2018). The following location includes implementing the model through program features on unseen data to test its accuracy. IEEE's article "Machine Learning in Compiler Optimization" (Machine Learning in compiler optimization, 2008) explains the importance of feature engineering for extracting static data structures extracted from the compiler's intermediate representation and dynamic profiling as properties or features before feeding into the models to remove any possible outliers.

J. Ansel, Y. L. Wong in 2011 explains the learning stage where, unlike other machine learning models, it generates its training data using transfer learning and compiler developer will select training programs for the teaching (J.Ansel, M.K., 2017). Then the developer complies using multiple optimization options, tunes the compiled binaries to analyze the best running time and then feed into the algorithm accordingly.

## **Deployment**

David I. August, the computer scientist at Princeton Institute (2003), talks about the challenges in deployment because of the variety of modern architectures and their complexities to run complex transformation algorithms as some might degrade performance on the running machine. Thus, it emphasized the need for compilers to employ predictive heuristics for optimization and predicting for making compiler sufficient to generate better code quality across various architectures up to the standard benchmark (World Medical Association, 2009). Zehn Wang explains the deployment process as simply inserting the data to analyze the best optimization conclusions on test data by first extracting the features from the first stage and then building a model on that approach (Wang, Z. and O’Boyle, M., 2018).

## **Criticism on thread coarsening**

Although thread coarsening is considered to provide high performance by increasing instruction-level parallelism and eliminating repetitive computations, Magni et al. explain its shortcomings and its adverse side effects, such as it can cause a stoppage in performance, reducing the total amount of parallelism and enhancing the register pressure (Cummins, C., Petoumenos, Magni et al.,2017). Also, thread coarsening is nontrivial as it depends on the target program and the architecture of the hardware it is running on.

## **Criticism on an iterative search**

In conventional methods, the iterative search was used for compiler optimization to overcome the performance bottlenecks. Still, Agakov and Thomson in 2006 (IEEE CGO-06) explain its drawback as the search time required to meet this performance criterion at each point is always the recompilation and re-executions of the complete programs. Toussaint. Furthers this by saying that although re-executions are acceptable in embedded code, the compilation cycles restrict the space of options being searched. Thus, in the bigger picture, it has a substantial barrier to adopting this approach. Cooper el al. examined the search space structure and devised new search-based algorithms that can outperform searching tasks, and one of the methods included predictive modelling (Wen, S., Chabbi, Cooper el al., 2017)

## **Shortcomings in the literature**

In the previous literature and conventional methods of Agakov about iterative search and Magin et al. about predictive heuristics, it has been seen that the predictive heuristics are incapable of anticipating the full effect of generating code optimization alternatives on final code quality (Georgiou, K., Blackmore, C., 2018,).  Spyridon Triantafyllis from Princeton University explains with a loop unrolling example. For every loop, the compiler would have to try numerous loop unrolling factors that aren't ideal in most cases. Heuristics can also not capture all instances like an exhaustive iterative compiler (Ashouri, A.H, Spyridon Triantafyllis., 2017).

The literature of the existing scholars made mention about these problems. However, it was unable to suggest an optimum and standardized solution that can work regardless of hardware architecture and can still comprise predictive heuristics salient features, which is why the need for Machine Learning in compiler optimization arrives (F. Agakov et al, 2006). Numerous machine learning models have been developed for this cause, including Genetic Algorithm (GA), supervised algorithms, unsupervised, k-means, clustering and regression models that works almost perfect even on redundant compilations and re-executions and overcome the shortcomings of previous literature (de Souza Xavier, 2018).

# Section 4. Research Methods

## **Research Method**

If we revisit what we discussed in the research question "To what context machine learning helps in finding improved compiler organization." To answer this question as it was the core of the research, I adopted qualitative research. The literature review in section 2 and the shortcomings in section 3, and the solutions to overcome those solutions was explained qualitatively instead of facts, figures, and quantifications (Patten, M.L. and Newhart, M., 2017). Qualitative research collects and analyses non-numerical data, which was followed in the study to address the 'how' and 'why' questions and prove them by literature reviews instead of numerical statistics.

## **Research Strategy**

Since the research method was qualitative, the research strategies were also qualitative. The data and information were gathered from the case studies, experimental results, their evaluation and analysis, content analysis of the researchers in the early century. These research methods helped determine the answers to the problems. They added value to the research by explaining the conventional techniques for optimizing the compiler. Still, we could also find out their complexities, limitations, and how the contributors suggested improving them (Barlow, J., Best, M., 2017). The case studies were vast and were based on the real-life problems to be related to our issues. I also used content analysis and experimental results from different algorithms and models to compare and contrast the process results with varying testing data to provide in-depth insights into the optimization problem.

## **Research Strategy – Data Collection**

The qualitative data collection method that I used was mainly exploratory as they are generally more focused on gaining the insights to determine the underlying reasons for a particular problem. This was also the primary data source because they were directly from research articles by IEEE and other journals. The other secondary data collection method was individual interviews of a few of my computer science professors who have expertise in machine learning, models, and compiler optimization. It was a face to face interview of 30 minutes each where I ensured to keep the domain of the questions limited and to ask open-ended questions that made it easier for professors to answer them in a detailed manner (Clark, K.R. and Vealé, B.L., 2018). I also conducted online surveys in the open-source community on Stack Overflow and GitHub. I got numerous responses from the questionnaire, which later I compiled and deduced my hypotheses of how machine learning helps in compiler optimization.

## **Data Analysis**

The content analysis method was used to comprehend both the verbal and behavioural data for summarizing and classifying the data.

## **Anticipated difficulties of the research**

Many difficulties were anticipated during the research, which included arranging a schedule for interviews. Due to online classes, most of the professors were teaching from different locations, and it was pretty challenging to request them to meet them for a short interview. I resolved this problem by visiting them for an interview rather than calling them to campus as it would be impossible for them. Another problem that I faced was a lack of relevant experience. Since the discussion of machine learning in compiler optimization is specific and I knew a lot more theory than practical, anticipating and deducing results from the questionnaires wasn't easy. It required a lot of research to understand them before making any conclusions from them.

## **Description of ethical issues**

As per the ethical code of COPE, "good research should be well adjusted, well-planned, appropriately designed, and ethically approved. To conduct research to a lower standard may constitute misconduct." (World Medical Association. 2009.) the interviewers were given the ethics form to agree on the roles of the research contributors, including matters of authorship and to protect their identity. Interviewers were only interviewed after signing the written consent form. This was for the secondary data collection (Maznun, M.D.B., Monsefi, R. and Nimehchisalem, V., 2017). For primary data collection, the ethical code included refraining from the copyright and giving proper credits to the contributors in case of citation or quotation from any research journal as that's not our source.

## **Validity/Reliability**

The primary data that we used for the research purpose was content analysis. They were authentic as published by IEEE and different universities, including Princeton University, verified by other departments. Thus, the primary data had no ambiguity. However, the secondary data gathering method was subjective, and the interviewee's experiences varied, which is why the data has been carefully selected. Only credible and relevant information was taken from the interviews and questionnaires.

# Section 4. Experiments

1. **Genetic programming (GP)** (feature engineering)

GP is used to optimize the priority functions associated with two well-known compiler **heuristics**: predicated hyper-block formation, and register allocation speeding up the system. For hyper-block selection, an average speed-up of *23%* & up to *73%* was observed while compilation. Compiler’s heuristic were tested over several benchmarks, the best general-purpose heuristic improves the predication algorithm by an average of *25%* on training set, and 9% on a completely unrelated test set. On average, system obtains a *6%* speedup when it specializes the register allocation algorithm for individual applications and the general-purpose heuristic for register allocation achieves a *3%* improvement.

1. **Clustering** (unsupervised learning)

Experiments for evaluating the effectiveness of the proposed approach address the exploration of optimization sequences in the context of the ReflectC compiler, considering 49 compilation passes while targeting a Xilinx MicroBlaze processor, and aiming at performance improvements for 51 functions and four applications. Experimental results reveal that the use of clustering-based DSE approach achieves a significant reduction in the total exploration time of the search space (*20×* over a Genetic Algorithm approach) at the same time that considerable performance speedups (*41%* over the baseline) were obtained using the optimized codes. Additional experiments were performed considering the LLVM compiler, considering 124 compilation passes, and targeting a LEON3 processor. The results show that our approach achieved geometric mean speedups of *1.49×*, *1.32×*, and *1.24×* for the best 10, 20, & 30 functions, respectively, and a global improvement of *7%* over the performance obtained when compiling with -O2.

# Section 5: References

Wang, Z. and O’Boyle, M., 2018. Machine learning in compiler optimization. *Proceedings of the IEEE*, *106*(11), pp.1879-1901.

Ashouri, A.H., Bignoli, A., Palermo, G., Silvano, C., Kulkarni, S. and Cavazos, J., 2017. Micomp: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning. *ACM Transactions on Architecture and Code Optimization (TACO)*, *14*(3), pp.1-28.

Lins, F.M., Tambara, L.A., Kastensmidt, F.L. and Rech, P., 2017. Register file criticality and compiler optimization effects on embedded microprocessor reliability. *IEEE Transactions on Nuclear Science*, *64*(8), pp.2179-2187.

Anderson, J.A., 2018. The pragmatics of audience in research and theory. In *The audience and its landscape* (pp. 75-94). Routledge.

Deng, C. and Namjoshi, K.S., 2018. Securing a compiler transformation. *Formal Methods in System Design*, *53*(2), pp.166-188.

de Souza Xavier, T.C. and da Silva, A.F., 2018. Exploration of compiler optimization sequences using a hybrid approach. *Computing and Informatics*, *37*(1), pp.165-185.

F. Agakov et al., "Using machine learning to focus iterative optimization," International Symposium on Code Generation and Optimization (CGO'06), 2006, pp. 11 pp.-305, doi: 10.1109/CGO.2006.37.

World Medical Association, 2009. Declaration of Helsinki. Ethical principles for medical research involving human subjects. *Jahrbuch Für Wissenschaft Und Ethik*, *14*(1), pp.233-238.

Mitra, S., Gupta, M.K., Misailovic, S. and Bagchi, S., 2017, February. Phase-aware optimization in approximate computing. In *2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)* (pp. 185-196). IEEE.

 Cummins, C., Petoumenos, P., Wang, Z. and Leather, H., 2017, September. End-to-end deep learning of optimization heuristics. In *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)* (pp. 219-232). IEEE.

Georgiou, K., Blackmore, C., Xavier-de-Souza, S. and Eder, K., 2018, May. Less is more: Exploiting the standard compiler optimization levels for better performance and energy consumption. In *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems* (pp. 35-42).

Wen, S., Chabbi, M. and Liu, X., 2017, April. Redspy: Exploring value locality in software. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 47-61).

Ashouri, A.H., Bignoli, A., Palermo, G., Silvano, C., Kulkarni, S. and Cavazos, J., 2017. Micomp: Mitigating the compiler phase-ordering problem using optimization sub-sequences and machine learning. *ACM Transactions on Architecture and Code Optimization (TACO)*, *14*(3), pp.1-28.

Patten, M.L. and Newhart, M., 2017. *Understanding research methods: An overview of the essentials*. Routledge.

Barlow, J., Best, M., Bohnenstengel, S.I., Clark, P., Grimmond, S., Lean, H., Christen, A., Emeis, S., Haeffelin, M., Harman, I.N. and Lemonsu, A., 2017. Developing a research strategy to better understand, observe, and simulate urban atmospheric processes at kilometer to subkilometer scales. *Bulletin of the American Meteorological Society*, *98*(10), pp.ES261-ES264.

Clark, K.R. and Vealé, B.L., 2018. Strategies to enhance data collection and analysis in qualitative research. *Radiologic technology*, *89*(5), pp.482CT-485CT.

Maznun, M.D.B., Monsefi, R. and Nimehchisalem, V., 2017. Undergraduate ESL students’ difficulties in writing the introduction for research reports. *Advances in Language and Literary Studies*, *8*(1), pp.9-16.

# Genetic Programming Applied to Compiler Heuristic Optimization Stephenson M., O’Reilly UM., Martin M.C., Amarasinghe S. (2003) Genetic Programming Applied to Compiler Heuristic Optimization. In: Ryan C., Soule T., Keijzer M., Tsang E., Poli R., Costa E. (eds) Genetic Programming. EuroGP 2003. Lecture Notes in Computer Science, vol 2610. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/3-540-36599-0_22>