MicroIREE: Multiple Dispatch Micro-Kernels

Alae-eddine Nasr[*]
School of Data Science
Chinese University of Hong Kong, Shenzhen
alae-eddinenasr@link.cuhk.edu.cn

Vincentius Janssen[*]
School of Data Science
Chinese University of Hong Kong, Shenzhen
vincentiusjanssen@link.cuhk.edu.cn

Yohandi[*]
School of Data Science
Chinese University of Hong Kong, Shenzhen
yohandi@link.cuhk.edu.cn

Hsu Wei-Chung[†]
School of Data Science
Chinese University of Hong Kong, Shenzhen
hsuweichung@cuhk.edu.cn

Abstract

This work details the optimization of matrix multiplication operations for the open-source ML compiler and runtime IREE through the utilization of multiple dispatch micro-kernels. We demonstrate the improvement in performance by performing training and inference of a ResNet50 MNIST model imported from TensorFlow compiled to run on our modified IREE runtime. Our results show a 300x performance improvement over running the same model with default settings on the stock IREE VM runtime. The primary contribution of this work is the incorporation of multiple dispatch for micro-kernels into the IREE VM runtime, resulting in up to a 74% speed-up over the stock IREE with only single dispatch capability.

1 Introduction

The increased adoption of machine learning (ML) for complex problem-solving and decision-making demands ever-increasing computational resources. The development of recent Large Language Models GPT-4 and Gemini is indicative of this trend: model performance scales with model size, suggesting that demand for computing will increase even further as we move towards larger models. This trend is in opposition with the projected decline of Moore's Law, a historical predictor of growth in the scaling of computing power. The increasing ubiquity of AI applications demands increasingly large amounts of computing while the cost and performance scaling of silicon processors is slowing down.

By optimizing model code for specific hardware architectures via optimization passes, compilers can significantly improve execution speed and reduce resource consumption, mitigating the compute limitations that constrain the development and deployment of SOTA models. This approach maximizes the potential of existing hardware, facilitating the continued evolution and use of ML models by ensuring that advancements can be sustainably scaled and integrated into a wide array of domains and applications.

At the heart of many ML workloads is matrix multiplication. This operation is computationally intensive and represents a significant performance bottleneck. AI architectures such as ResNet rely heavily on matrix multiplication to execute convolution layers for image feature extraction, while the Transformer architecture uses them within the attention mechanism to calculate relevant sequences [1][2]. Hence, optimizing the performance of matrix multiplication directly impacts the overall speed

[*]Equal contributions
[†]Research advisor

and efficiency of these models. Improvements in this area translate to faster model training, reduced inference times, and an ability to handle larger, more complex models within the same computational constraints.

We sought to combine and compound the benefits of both by improving the performance of matrix multiplication of the IREE ML compiler, which was achieved by utilizing a matrix multiplication micro-kernel in the runtime for a 178x speed-up on ResNet50 inference. This work further extends the performance gains by enabling multiple dispatches of micro-kernels in the VM runtime backend by unrolling the dispatcher loop, resulting in a further speed-up of 74% for a total performance gain of over 300x.

2 Related Work

2.1 IREE

IREE (Intermediate Representation Execution Environment) is an MLIR-based end-to-end compiler and runtime that lowers Machine Learning (ML) models from various frameworks such as TensorFlow, PyTorch, and JAX to a unified IR that scales from edge to datacenter deployments [3].

IREE supports different ML frameworks via an importer that converts functions into an MLIR form. The IREE compiler uses MLIR to progressively lower an ML program into a modular code that runs on the IREE runtime. MLIR dialects are key to this lowering process. Dialects are collections of operations and types that represent a program, and different dialects can be extended from the base MLIR operation set to model different levels of program abstraction.

The different dialects utilized by IREE allow easier development and implementation of optimization passes. The TOSA (Tensor Operator Set Architecture) models the ML program as a graph of tensor operations, facilitating graph reordering. The Linalg dialect, meanwhile, represents nested loop computation, making it easy to implement loop fusion, tiling, and interchange optimizations. The Vector dialect models virtual vectorized operations that facilitate vectorization, and the LLVM dialect enables straightforward translation of the resulting MLIR into LLVM-IR for compilation into a binary module using the LLVM compiler toolchain.

IREE modules are a container format for an interpretable bytecode or a compiled binary, depending on the target backend selected during compilation. The runtime runs compiled binaries through the use of a Hardware Abstraction Layer (HAL) that interfaces with target hardware and interprets bytecode in a Virtual Machine (VM).

Naturally, running compiled binaries would give a performance that far exceeds that of interpreted bytecode. Our benchmark ResNet50 model runs over 400x slower interpreted on the stock IREE VM runtime compared to a binary compiled version running via the IREE CPU HAL runtime.

3 Approach

Our optimization approach focused on improving the performance of matrix multiplication operation, a computationally expensive operation that is a component of many model architectures.

From initial profiling results, we found that matrix multiplication was extremely costly when compiled for the VMVX target. Model inference latency was over 400x higher when running interpreted on the stock IREE VM runtime compared to a binary compiled version of the same model running on the IREE CPU HAL runtime. A surprising result was that it was running even slower than the original model on TensorFlow.

Lack of loop unrolling, tiling, and fusion was found to be the culprit of performance deficiency, which was rectified by enabling a compilation flag on the IREE compiler.

Further analysis of the VM runtime presented another opportunity for enhancement. The VM runtime was found to dispatch micro-kernels one at a time. In a compiled binary, this would not be a big issue as the compiler would unroll the loop, but having it in the runtime precludes the chance for optimization as the data dependency is unknown. Hence, we manually unroll the VM runtime dispatcher to issue a block of 4 micro-kernels at a time to increase ALU utilization.

4 Experiments

4.1 Data

The dataset utilized in this study is the Modified National Institute of Standards and Technology (MNIST) dataset, a widely recognized benchmark in the field of machine learning. MNIST consists of a collection of 28x28 pixel grayscale images of handwritten digits (0 through 9), totaling 70,000 examples, with 60,000 images for training and 10,000 for testing [4]. Originating from the National Institute of Standards and Technology (NIST), the dataset has been modified and curated for research purposes. LeCun et al. introduced MNIST in their seminal work "Gradient-Based Learning Applied to Document Recognition" (1998), and it has since served as a standard benchmark for evaluating various machine learning algorithms, particularly those related to image classification tasks.

4.2 Evaluation method

We use ResNet-50, a commonly deployed model architecture, to evaluate the speed-up obtained from our modifications to the IREE runtime. ResNet-50 is a Convolutional Neural Network (CNN), which relies heavily on convolutional layers typically implemented as matrix multiplication operations.

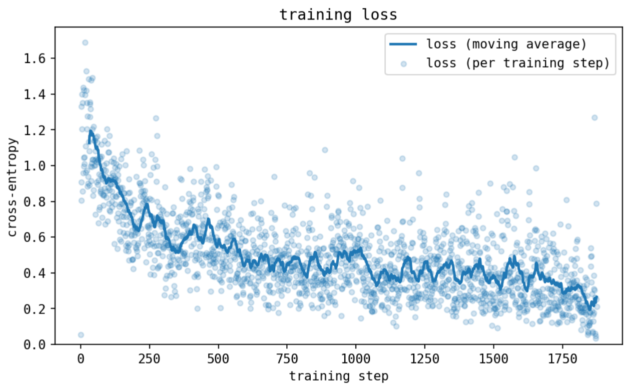
Two key metrics are used to measure performance: inference latency and training latency. Inference latency refers to the time taken by the model to make predictions on new, unseen data while training latency pertains to the time required to train the model on the training dataset. These metrics can be used to determine the efficiency and practicality of model deployment with our compiler infrastructure. Specifically, we focus on evaluating inference latency as it directly impacts real-world usability since models are inferenced far more often than they are trained. Lower inference latency also translates to faster model responses, making it crucial for applications where speed is critical.

We test our optimized model on both ARM and x86 architectures, ensuring that our improvements are platform-agnostic.

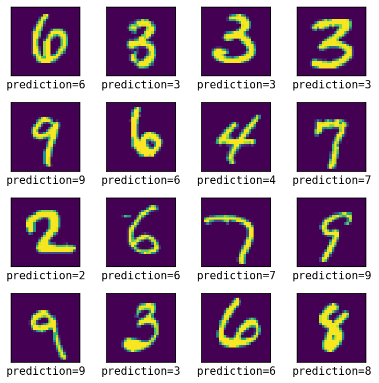
4.3 Technical correctness

To ensure the technical correctness of our model, we performed thorough training and verification of our approach.

As previously mentioned, our model was trained on a dataset of 70000 samples, with approximately 80% of the data used for training and approximately 20% for validation. We employed a cross-entropy loss function as an optimizer and trained the model for 2000 training steps. To validate our model's correctness, we visualized the predictions on a random selection of held-out data. The following figure shows example images with their corresponding predictions, demonstrating the model's ability to accurately classify handwritten digits.



(a) Training Loss



(b) Sample Predictions

Figure 1: Training Loss and Sample Predictions

4.4 Results and Analysis

This section presents quantitative results demonstrating the significant performance improvements for the VMVX target achieved by implementing micro-kernel multiple dispatch in the VM runtime backend. We compare these results against ResNet-50 performance on the stock IREE VM runtime (interpreted), TensorFlow, and IREE HAL runtime compiled using LLVM with best compilation flags. The results are gathered from runs on two different micro-architectures on two ISAs: x86_64 and ARM AArch64.

4.4.1 Inference Latency Improvement on x86_64

The implementation of a multiple dispatcher in the VM runtime backend has led to a remarkable performance improvement in inference latency on Golden Cove, reducing the time from 1.03 ms to 589 µs, which corresponds to an enhancement of almost 75%. This improvement significantly enhances compute efficiency and responsiveness, which is extremely relevant for real-time applications.

4.4.2 Performance Comparison to TensorFlow and LLVM

TensorFlow's inference latency on Golden Cove is 2.06 ms, which is substantially higher than the IREE VMVX target with a multiple dispatch micro-kernel at 589 µs. This represents a performance enhancement of more than three-fold over TensorFlow.

Comparatively, LLVM exhibits an inference latency of 421 µs on Golden Cove, which is slightly lower than IREE VMVX's 589 µs. However, the gap is narrow, demonstrating that VMVX is highly competitive with LLVM's optimized configurations.

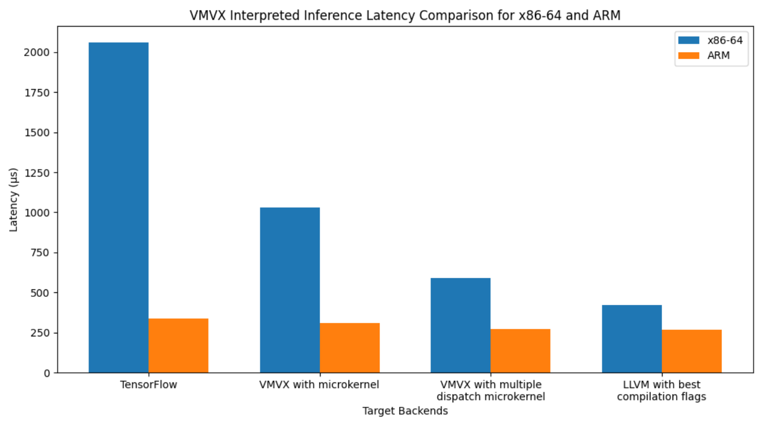
4.4.3 ARM Architecture Performance

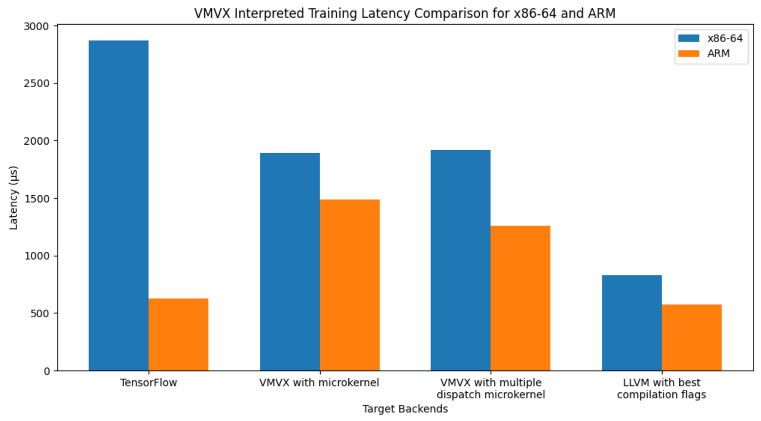
On the Firestorm micro-architecture, the VMVX with a multiple dispatch micro-kernel shows an inference latency of 272 µs, very close to LLVM's 268 µs, and significantly outperforms TensorFlow's 339 µs. This illustrates the effectiveness of multiple dispatch even on a much more powerful system.

4.4.4 Training Latency Analysis

The training latency of ResNet-50 on VMVX with multiple dispatch micro-kernel on Golden Cove is 1.92 ms, compared to TensorFlow's 2.87 ms and LLVM's 831 µs. While LLVM maintains a lead in this area, the improvement in the VMVX target performance over the single dispatch runtime, which recorded 1.89 ms, is notable.

On Firestorm, the multiple dispatch micro-kernel modification reduces the training latency to 1.26 ms from 1.49 ms observed in the single dispatch setup, demonstrating a consistent improvement across different operations and micro-architectures. These results underscore the substantial advancements made in the IREE VM runtime through the implementation of micro-kernel multiple dispatch, which enables running an interpreted model competitively against the same model compiled.





From the results of our experiments, we can see that implementing multiple dispatch of micro-kernels in the IREE VM runtime is highly effective. We achieve performance gains of over 70% versus single dispatch micro-kernels for the VMVX target ResNet-50 inference as measured on the Intel Golden Cove micro-architecture. We find that the performance increase of dispatching multiple micro-kernels is not as large on the Apple Firestorm micro-architecture, at only 10-15%. This outcome is expected due to the faster memory subsystem (2x bandwidth), larger cache (320KB vs. 80KB L1 per core), and wider core (decode and execution) which collectively minimize the time spent waiting for and executing instructions.

5 Conclusion

We have demonstrated the transformative impact of integrating multiple dispatch of micro-kernels into the IREE VM runtime. Our approach led to a dramatic performance improvement in both inference and training latency of ResNet-50 for the VMVX target, showcasing the ability of our optimized runtime interpreter to compete with established machine learning frameworks such as TensorFlow.

Through rigorous testing and evaluation, we observed a near 75% improvement in inference latency on Intel's Golden Cove micro-architecture, highlighting the effectiveness of multiple dispatch micro-kernels in enhancing computational efficiency. Although the improvements on the Firestorm micro-architecture were less pronounced, the enhancements were still found to be effective across different hardware platforms.

The project provided us insight into optimizing matrix multiplication, a fundamental operation for many AI models. We learned to better utilize hardware to enhance runtime performance.

However, we have found limitations as well. Notably, the advantages of our multiple dispatch strategy were less significant on wide micro-architectures such as Apple Firestorm, as it has significantly more memory bandwidth, cache, and a wider-core versus Golden Cove. This finding underscores the importance of understanding and adapting to the nuances of hardware architectures when optimizing machine learning operations. Furthermore, training latency seems to be worse than on TensorFlow.

The enhancements implemented in the IREE VM runtime through the introduction of multiple dispatch micro-kernels represent a significant advance in the performance optimization of IREE. The ongoing refinement and expansion of these techniques hold the promise to further elevate the capabilities of the IREE compiler infrastructure to meet the escalating demands of modern AI applications.

6 Future Work

Moving forward, there are several promising directions for extending the capabilities of the IREE VM runtime:

- **Just-In-Time (JIT) Compilation:** Implementing JIT compilation in the backend can dynamically optimize machine code at runtime. This adaptation could potentially mitigate some limitations observed with hardware architectures like Apple Firestorm.
- **Hardware-specific Optimizations:** Deeper investigations into optimizations tailored specifically for new hardware architectures can additional performance gains. This focus is increasingly pertinent as new hardware technologies continue to emerge.
- **Broadening Optimization Scope:** While this project concentrated on matrix multiplication, other computationally intensive operations in AI workloads could also benefit from new micro-kernels. Exploring these opportunities could lead to comprehensive improvements across other aspects of model training and inference.
- **Enhancing VMVX Architecture:** Additional enhancements in the VMVX dialect itself could further improve performance. Potential improvements could include optimizing graph reordering and creating new dialects for optimization passes. Such developments would not only improve execution efficiency but also reduce runtime overhead.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *Advances in Neural Information Processing Systems*, 30, 2017.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[3] Liu, Hsin-I Cindy and Brehler, Marius and Ravishankar, Mahesh and Vasilache, Nicolas and Vanik, Ben and Laurenzo, Stella. TinyIREE: An ML Execution Environment for Embedded Systems From Compilation to Deployment. *IEEE Micro*, 42(5):9–16, 2022.

[4] Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Appendix

Table 1: Latency Measurements for Tensorflow and VMVX Target (Interpreted)

	Intel Golden Cove		**Apple Firestorm**	
Target Backends	Inference (ms)	Training (ms)	Inference (ms)	Training (ms)
Tensorflow	2.060 ± 0.360	2.870 ± 0.436	0.3390±0.0851	0.624 ± 0.199
VMVX without micro-kernel	184.000 ± 0.687	322.000 ± 0.853	103.000 ± 0.667	193.00 ± 6.96
VMVX with micro-kernel	1.030 ± 0.131	1.890 ± 0.123	0.3100±0.0115	1.490 ± 0.312
VMVX with multiple dispatch micro-kernel	0.5890±0.0908	1.920 ± 0.194	0.272 ± 0.003	1.260 ± 0.506

Table 2: Latency Measurements for Different Compilation Flags on LLVM Target (Compiled)

	Intel Golden Cove		**Apple Firestorm**	
Target Backends	Inference (ms)	Training (ms)	Inference (ms)	Training (ms)
LLVM []	0.4210±0.0138	0.8310±0.0115	0.3530±0.1390	0.612 ± 0.028
LLVM [–iree-llvmcpu-enable-ukernels=all]	0.8670±0.1500	1.830 ± 0.332	0.3400±0.0215	0.608 ± 0.037
LLVM [–iree-llvmcpu-loop-interleaving]	0.4240±0.0346	0.8340±0.0239	0.3070±0.0074	0.678 ± 0.104
LLVM [–iree-llvmcpu-enable-ukernels=all, –iree-llvmcpu-loop-unrolling]	0.7690±0.0597	1.3100±0.0575	0.2940±0.0918	0.574 ± 0.023
LLVM [–iree-llvmcpu-loop-interleaving, –iree-llvmcpu-loop-unrolling]	0.4270±0.0169	0.862 ± 0.055	0.2990±0.0296	0.580 ± 0.030
LLVM [–iree-llvmcpu-enable-ukernels=all, –iree-llvmcpu-loop-interleaving, –iree-llvmcpu-loop-unrolling]	0.7330±0.0607	1.3100±0.0683	0.345 ± 0.036	0.885 ± 0.247
LLVM [–iree-llvmcpu-loop-vectorization]	0.4350±0.0171	0.8450±0.0111	0.2970±0.0203	0.600 ± 0.028
LLVM [–iree-llvmcpu-enable-ukernels=all, –iree-llvmcpu-loop-vectorization]	0.7530±0.0853	1.4100±0.0559	0.3270±0.0521	0.712 ± 0.145
LLVM [–iree-llvmcpu-loop-interleaving, –iree-llvmcpu-loop-vectorization]	0.4570±0.0302	0.8290±0.0095	0.2680±0.0174	0.571 ± 0.031
LLVM [–iree-llvmcpu-enable-ukernels=all, –iree-llvmcpu-loop-interleaving, –iree-llvmcpu-loop-vectorization]	0.8040±0.0925	1.5000±0.0646	0.289 ± 0.022	0.573 ± 0.025
LLVM [–iree-llvmcpu-loop-unrolling, –iree-llvmcpu-loop-vectorization]	0.4430±0.0196	0.8570±0.0132	0.3310±0.0797	0.626 ± 0.028
LLVM [–iree-llvmcpu-enable-ukernels=all, –iree-llvmcpu-loop-unrolling, –iree-llvmcpu-loop-vectorization]	0.7660±0.0535	1.480 ± 0.228	0.2850±0.0297	0.744 ± 0.342
LLVM [–iree-llvmcpu-loop-interleaving, –iree-llvmcpu-loop-unrolling, –iree-llvmcpu-loop-vectorization]	0.4300±0.0306	0.8400±0.0447	0.3230±0.0421	0.751 ± 0.132
LLVM [–iree-llvmcpu-enable-ukernels=all, –iree-llvmcpu-loop-interleaving, –iree-llvmcpu-loop-unrolling, –iree-llvmcpu-loop-vectorization]	0.7520±0.1250	1.2800±0.0285	0.4370±0.0505	0.780 ± 0.269

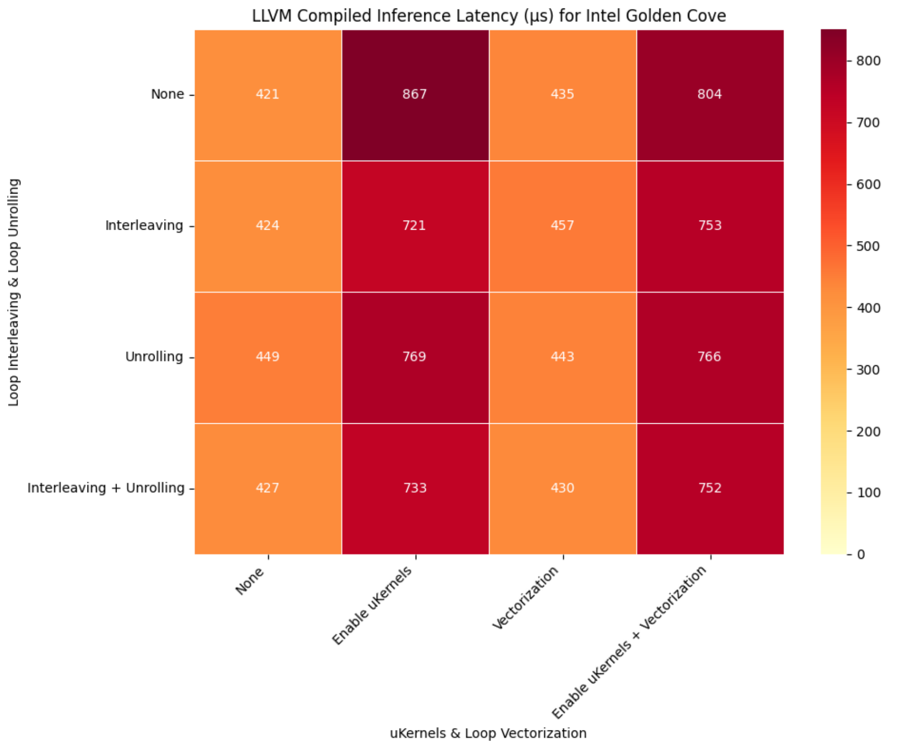


Figure 2: LLVM Compiled Inference Latency (μs) for Intel Golden Cove

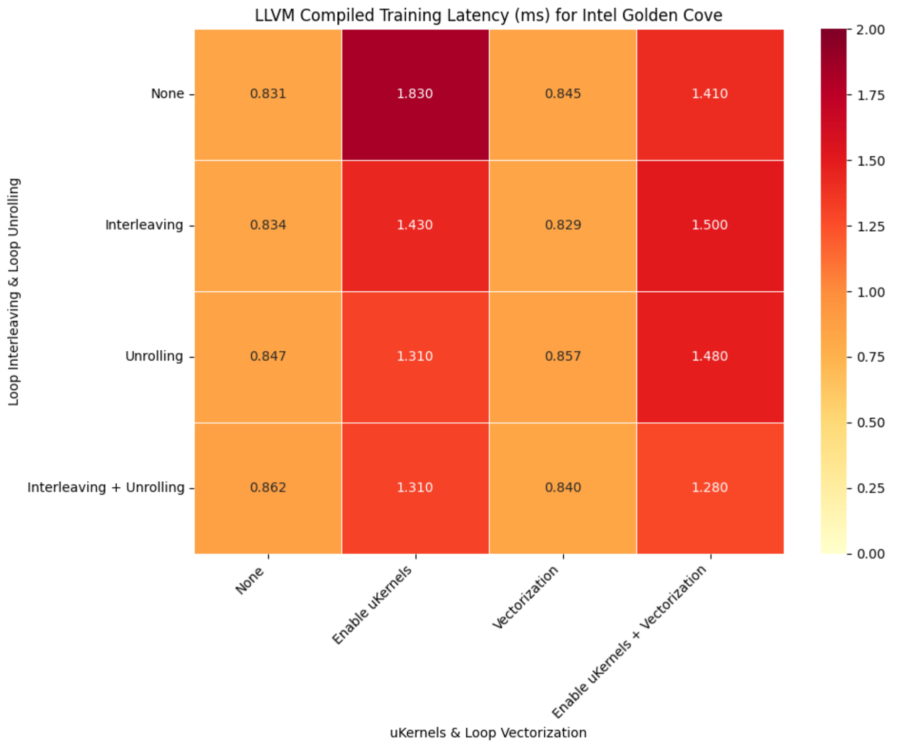


Figure 3: LLVM Compiled Training Latency (ms) for Intel Golden Cove

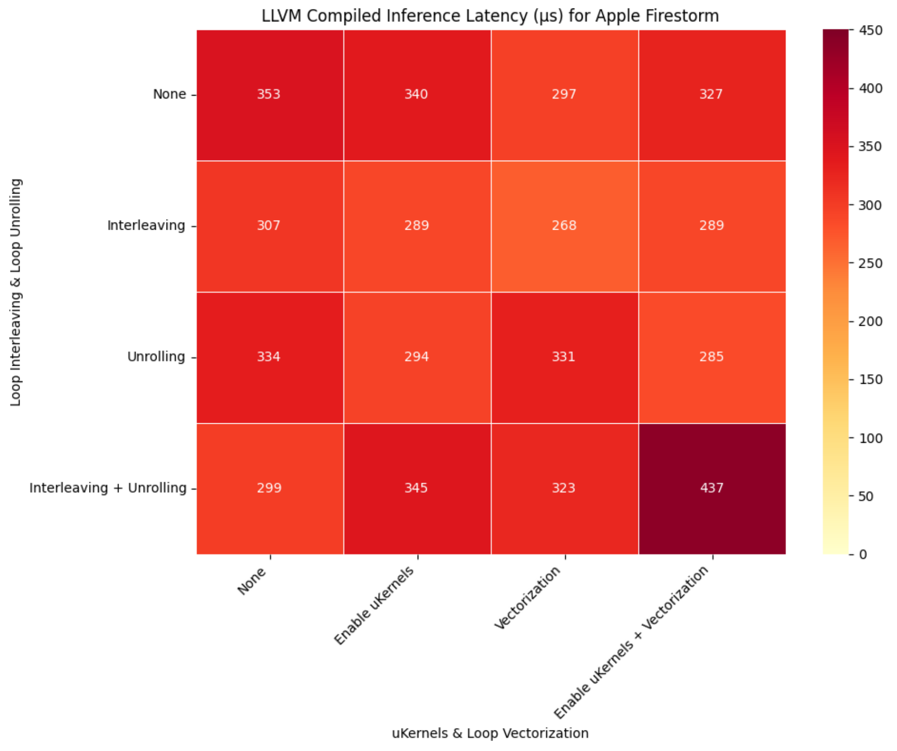


Figure 4: LLVM Compiled Inference Latency (µs) for Apple Firestorm

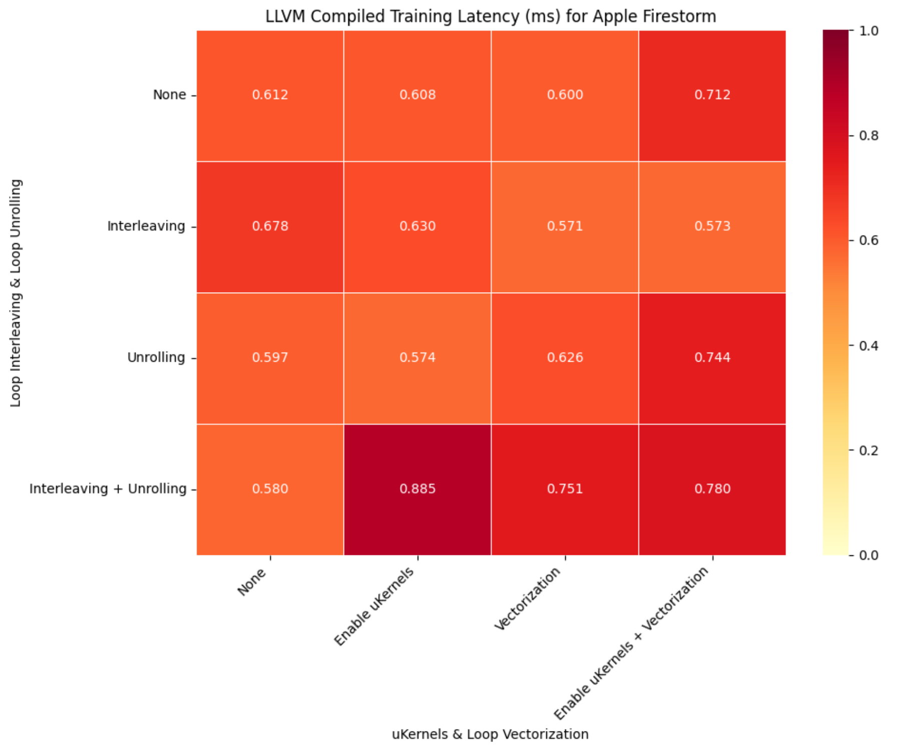


Figure 5: LLVM Compiled Training Latency (ms) for Apple Firestorm