2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 15-16, 2024

Compiling AI Workloads for On-Device Inference on Heterogeneous Systems using MLIR

Jan Moritz Joseph, RooflineAI GmbH, Aachen, Germany (joseph@roofline.ai)

Maximilian Bartel, RooflineAI GmbH, Aachen, Germany (bartel@roofline.ai)

Rainer Leupers, RWTH Aachen University, Aachen, Germany (leupers@ice.rwth-aachen.de)

I. Abstract

AI workloads can be found in nearly all application areas. Especially the edge-AI market is rapidly evolving both on the application and the hardware side. Dedicated hardware solutions for on-device inference have been proposed so that different chips and IPs are sold to address varying customer use cases. In many cases, a dedicated software solution programs the AI workload to the target. As platforms are more complex and are inherently heterogeneous, a flexible deployment solution based on a retargetable compiler is required. We introduce Roofline's Software Development Kit (SDK) that employs different intermediate representations (IRs) to lower the AI model to an abstraction that can target the desired hardware. This enables both retargetability and support for heterogeneous components. It outperforms state-of-the-art solutions in efficiency and enables up to 15x memory savings and up to 5x latency reduction for current workloads on embedded systems like a Raspberry Pi.

II. Introduction

Today, AI is used in nearly all application domains. It has vitally contributed to innovation, ranging from computer vision, over natural language processing, to creative generation. The rise of AI is accompanied by a rise of required compute resources, mainly on server farms. This has driven the power and resource consumption towards new heights, but it also limits the use of AI to network-connected systems with recurring costs.

AI models used for inference can be found in many sizes: From large models run on server or cloud systems down to orders-of-magnitude smaller solutions run on the edge. Since server costs are accumulating and network connectivity is not guaranteed, especially the edge-AI market is rapidly evolving both on the application and the hardware side. On the application side, AI models are constantly changing and need to be deployed to billions of devices (Thürer 2023). These models can bring ever larger capability into smaller resource footprints, which shifts the limits of edge AI from on-chip memory size to on-chip compute capabilities. On the hardware side, solutions develop fast with novel architectures and technologies both using conventional architecture improvements (Ananda Samajdar 2021) or novel manufacturing technologies such as 3D integration (Jan Moritz Joseph 2021).

Dedicated hardware solutions for on-device inference have been introduced widely into the market. They provide the basis for energy-saving solutions to execute growing AI models locally. Recently, many chip vendors started integration of neural processing units (NPUs) or AI-focused GPUs along their CPU lineups. This trend can be found at any scale, from desktop systems down to microcontrollers. Examples are Intel's Core Ultra products with an AI engine, AMD's Ryzen cores with Ryzen AI, or NXP's i.MX family with the eIQ Neuron NPU (NXP 2024). Especially in the edge-AI market, a diverse set of target architectures provides either specialized or generalized approaches. Different chips and IPs are sold to address varying customer use cases and are being integrated into numerous heterogeneous platforms.

In many cases, a dedicated software tooling is shipped to map the AI workload to the target hardware. Often, this software is based on legacy solutions (like Tensorflow Lite) that do not support every new feature from the fast-moving novel AI models such as large language models (LLMs). In other cases, the solutions are specialized for niche applications and interactions with other system components are custom.

This situation yields a deployment and integration challenge during product development both on the AI user and the chip vendor side: The AI users struggle with the legacy software, due to useability and coverage challenges, e.g., costly on-boarding times and limited possibilities to execute all their models. The chip vendors need to integrate mainly hand-written kernels into library-based software, which wastes engineering hours and limits performance. Therefore, a more flexible and more performant solution is required. It can be found in retargetable AI compilers, as we demonstrate here.

This paper introduces Roofline's retargetable AI compiler technology that overcomes the adoption challenges of novel AI workloads for novel AI acceleration hardware. Specifically, we contribute the following:

1. We show the flexibility of our solution to adopt different use cases and target hardware by demonstrating the retargetability using intermediate representations from MLIR.
2. We show the efficiency of our solution by exploring different benchmarks and target hardware and comparing them against state-of-the-art solutions.
3. We show the ease of use in benchmark case studies for a concrete edge AI deployment.

III. Related Works

For local deployment of AI models on resource-constrained systems, Tensorflow Lite (TFLite) (Tensorflow 2024) is the most common solution. It can be used from mobile systems, like smartphones running Android, down to small microcontrollers. It is mostly integrated into chip vendor Software Development Kits (SDKs), like ST Micro's Edge AI tools (Tensorflow 2024). For AI models, TFLite is compatible with many Tensorflow models and offers quantization. However, it does not support Pytorch models. Running current models like LLMs is not possible due to the lack of a library supporting dynamic shapes and KV caching. TFLite executes the model via library calls on a host CPU. In addition, it offers a software interface called *delegates* that allows the integration of custom kernel code to support dedicated AI accelerators.

Another solution is TVM. TVM is an AI compiler that offers support to include custom kernel code for dedicated AI accelerators. TVM offers a variant for embedded systems, microTVM, that can also run in bare metal mode (Apache 2024). It is more flexible than TFLite, as it offers a fully compiled solution. Like TFLite, custom accelerators can be included. They are not linked from a library call but by identification of a matching intermediate representation (IR) in the compiler stack.

ExecuTorch is a new solution for on-device inference across mobile and edge devices including wearables, embedded devices and microcontrollers (The Linux Foundation 2024). It competes with TFLite for Pytorch models. It offers ahead-of-time compilation and a runtime to different systems. It lowers the neural network graph to an IR called Aten dialect and decomposes complex operators into more common ones. This is afterwords further lowered to an even more constrained Edge dialect and then lowered to the Backend dialect. After AOT memory planning the generated program is serialized into a flatbuffer. The ExecuTorch runtime, similarly to TFLite, takes the serialized application code as well as a library of custom kernels to execute the neural network on the devices. ExecuTorch is currently in alpha and only supports PyTorch models. While it can support models with dynamic shapes as well as AOT memory planning, it is still required to provide a set of handwritten/pre-generated kernels to support target platforms.

For specific chips, such as in the field of neuromorphic computing, there are full-custom software solutions, e.g. Intel Lava for the Loihi platform (Intel Inc. 2024) or Innatera's Talamo SDK (Innatera Nanosystems BV 2024). These software solutions are tailored towards specific target software and hardware and therefore are not generally applicable to enable programming of AI on edge devices.

MLIR, part of the LLVM ecosystem, is a compiler framework that offers a set of pre-defined IRs tailored to AI workloads (LLVM Project 2024). One example is the TOSA IR that encapsulates typical hardware abstractions like conv layers. Another example is the Linalg, which represents linear algebra operations (LLVM Project 2023). MLIR allows to build retargetable AI compiler technology via custom defined IRs. One open-source compiler using MLIR is IREE (The IREE Authors 2024). It targets server-scale systems with backends for CPUs and GPUs, like AMD and Nvidia cards. One variation of IREE for embedded systems is TinyIREE, which proved its technical feasibility, but is not yet integrated into products to the best of the authors' knowledge (Liu, Brehler et al. 2022).

To summarize the previous related works, there is not yet a unified solution for on-device execution of AI models targeting energy-efficient and heterogeneous systems on the edge. MLIR offers a suitable infrastructure, and the IREE compiler technology can be used as a baseline to optimize for non-cloud use cases. None of the existing solutions offers flexible, efficient, and simple design flows, though. Roofline contributes such software with its novel SDK that comprises the *attic* frontend and the *cellar* backends for different devices. We will show in this paper how to apply our technology for heterogeneous systems.

IV. ROOFLINE SDK

The key challenges in AI programming are flexibility, efficiency, and simplicity to bring customer models to devices and reduce the time to market:

1. A *flexible* AI SDK refers to the capability to program different AI models to varying hardware targets. It should be able to use AI models from any relevant AI modeling framework, e.g., Tensorflow, Pytorch, and ONNX. It should also cover all relevant current and future AI model types, e.g., CNNs, RNNs, Transformers, or LLMs. Regarding hardware, it should target most of the relevant systems, such as CPUs, GPUs, and NPUs.
2. An *efficient* AI SDK reduces the resource footprint during execution. Relevant metrics are the live memory consumption during an inference run, the latency of the calculation of a single inference run (including communication times from main memory to CPUs and GPUs), and the size of the compiled artifact.
3. A *simple* AI programming software hides the complexity of heterogeneous systems from the user and allows non-experts in the domain of embedded systems and compiler developers to bring their AI workloads to the best-suited target device.

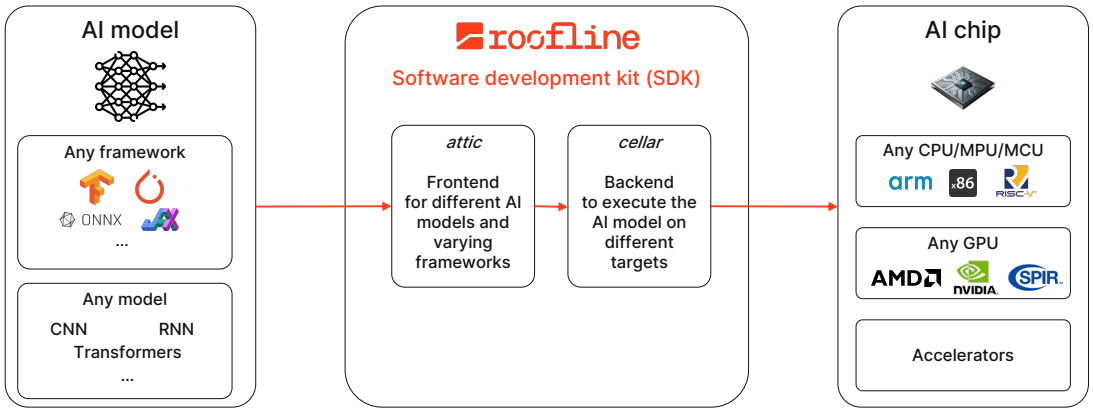

Figure 1: RooflineAI's SDK enables AI deployment from any relevant framework to systems including heterogeneous components.

Roofline's SDK addresses these three challenges with its innovative retargetable compiler technology. It is the one-stop deployment solution for any AI model to any target hardware. Its capabilities are summarized in Figure 1. The AI users load their model into Roofline's SDK. It is read by a frontend (*attic)* that supports any standard AI modeling framework (Pytorch, Tensorflow, TFLite, ONNX, JAX). Via different IRs, *attic* lowers the AI model to an abstraction level suitable for the target hardware. One example are optimizations on a graph level, which are linked to individual kernels, e.g., for a convolution. In another example on the lower levels, *attic* emits an IR used in a backend for automated code generation. This code generation for the target hardware is done by Roofline's *cellar* backends. The users can select the best-suited backend for their case depending on the components found in their hardware platform. The *cellar* backend also includes a runtime environment, executed on a CPU within the system. It manages the AI model execution and offers interfaces to the different targets that do the heavy lifting of all AI model calculations.

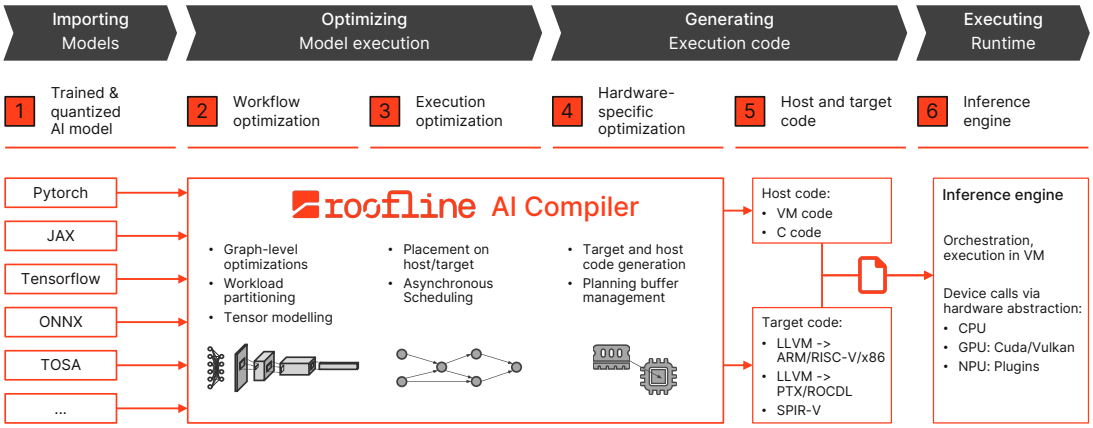

Figure 2: Schematic view of the tool flow in RooflineAI's SDK with different hardware support.

The detailed view of the compilation toolchain is shown in Figure 2.

In step 1, a trained AI model is read. We provide a Python interface that can be easily used next to Pytorch, Tensorflow, etc.

In step 2, we optimize the model on different levels of abstraction. As our compiler technology is based on MLIR, an open-source compiler framework within the LLVM ecosystem, we can utilize its standard IRs, e.g., TOSA on the graph level, or Linalg on the level of linear algebra operations for all relevant optimizations. We start with graph-level optimizations to improve the execution patterns in the whole neural network. Next, we partition the workload so that it fits the target hardware; for a heterogeneous platform, each partition fits the heterogeneous components' sizes. We lower the linear algebra IR to model the tensors and find the best tensor calculation scheme. The modular approach in step 2 enables targeting heterogeneous platforms by identifying the optimal operations per target and retargeting the corresponding IR to the backend.

In step 3, we optimize the execution. We coordinate the communication between the target and the host. The host refers to our runtime, executed on one CPU core that acts as a central controller and orchestrates the whole execution, e.g., by memory management. The runtime also enables asynchronous scheduling between different system components by monitoring the data flow and including barriers to synchronize buffers.

In step 4, target binary code is generated for the corresponding hardware. As stated, we support CPUs (ARM, RISC-V, x86), GPUs, or dedicated AI accelerators. Here, we generate the target code for each operation mapped to a target device. For the example of a CPU target, we lower an operator code to LLVM IR and compile it further with LLVM; for the example of a GPU target, we lower the operator code to PTX or SPRI-V.

In step 5, the heterogeneous system is planned. The host code contains all operations for our runtime that orchestrates the overall execution. The target code contains all operators for each target. These are brought together in multiple object files that are passed to the user for execution on the system.

In step 6, at runtime, an inference engine runs on the host CPU of the system and schedules the workload to the different targets. By extending the runtime via an abstract device interface we enable plug-and-play integration of different targets.

V. HETEROGENEOUS PLATFORMS

Heterogeneous platforms can be found on many SoCs. Most SoCs used in edge applications contain one or more CPUs, a GPU, DSPs, and more frequently, NPUs for AI workloads. Many of these NPUs are tailored for a specific application area, such as supporting popular layer shapes found in CNNs, RNNs, LSTMs, or Transformers. This yields high performance for this specific workload type but limits the portability of the SoC towards newer algorithms.

Roofline's SDK fully supports heterogeneous SoC platforms with different components. With its modular structure relying on MLIR dialects, we can select the best-suited one for each component. For example, we use the graph-level dialects to offload to fixed-function accelerator targets with vendor-specific software libraries, or we go through lower-level IRs (linalg and affine) to lower to LLVM IR to target a CPU. For GPUs, we rely on NVVM IR (NVIDIA Corporation & affiliates 2012) and rocdl IR (LLVM Project 2022).

During the ahead-of-time compilation, we allow the user to manually select a target and provide feedback if it does not support the entire network. In such cases, we identify access code and compile it for the CPU inside the SoC as a fallback solution. This approach enables us to provide code coverage beyond the SoC's capabilities with its fixed-function accelerators. We also offer heuristics to select the best target. We sum estimated memory access times and compute latencies for tiled workload parts and select the optimal combination. Improved heuristics that use vendor-specific performance models are currently under development with partners.

After identifying the best target, we either use the function library provided for fixed-function units or compile kernels for the target. For CPUs, we tile to fit cache sizes, vectorize, bufferize, emit LLVM IR, and link via LLVM. For GPUs, we distribute the workload among work groups, target integer, tensor, or vector units, bufferize, distribute the threads, and emit nvvm/rocdl IR. We also support computing-in-memory modules by tiling the workload and distributing before bufferization (Pelke, Bosbach et al. 2023).

VI. Results

We compare the efficiency of our solution to TFLite (for Tensorflow models) using a Pixel 8 Pro mobile platform, and TorchInductor (for Pytorch models) using a Raspberry Pi 5. We measure the inference latency and its required memory usage during execution. We selected a set of relevant benchmarks covering Transformers and CNNs. The results are plotted in Figure 3.
They show that we outperform TFLite and TorchInductor in both dimensions. We are up to 5x faster and up to one order of magnitude more memory-saving than TorchInductor. TorchInductor is optimized for cloud applications, in which these two metrics are less relevant than in the edge AI space. Looking at TFLite, we also outperform this solution, but with a smaller margin, as both solutions are optimizing for edge AI. Please note that TFLite does not support LLMs, while we do cover also these networks.

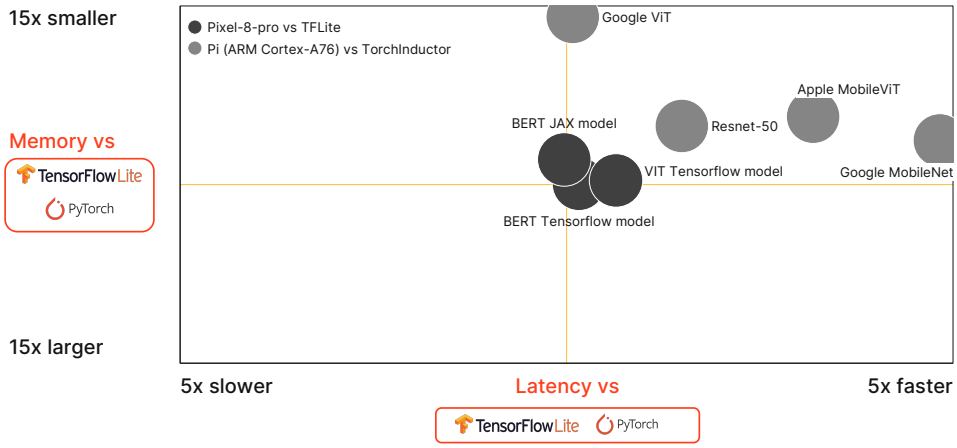

Figure 3: Comparison of the efficiency of the roofline SDK with competing technologies.

VII. Conclusion

To summarize, Roofline's SDK proves its advantages for smaller systems, while each deployment is done with two lines of code in Python using our imported library. Compared to other solutions, this is a useability advantage, since many other frameworks imply longer time-to-market for edge AI solutions wasting precious engineering hours.

VIII. References

Ananda Samajdar, J. M. J., Yuhao Zhu, Paul Whatmough, Matthew Mattina, Tushar Krishna (2021). A systematic methodology for characterizing scalability of DNN accelerators using SCALE-sim. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), IEEE.

Apache (2024). "microTVM: TVM on bare-metal." from https://tvm.apache.org/docs/topic/microtvm/index.html.

Innatera Nanosystems BV (2024). "Talamo Software Development Kit." Retrieved June 21, 2024, from https://innatera.com/products/talamo-software-development-kit.

Intel Inc. (2024). "Lava A Software Framework for Neuromorphic Computing ". Retrieved June 21, 2024, from https://github.com/lava-nc/lava.

Jan Moritz Joseph, A. S., Lingjun Zhu, Rainer Leupers, Sung Kyu Lim, Thilo Pionteck, Tushar Krishna (2021). Architecture, Dataflow and Physical Design Implications of 3D-ICs for DNN-Accelerators. 22nd International Symposium on Quality Electronic Design (ISQED), IEEE.

Liu, H.-I. C., et al. (2022) TinyIREE: An ML Execution Environment for Embedded Systems from Compilation to Deployment.

LLVM Project (2022). "'rocdl' Dialect." Retrieved July 1, 2024, from https://mlir.llvm.org/docs/Dialects/ROCDLDialect/.

LLVM Project (2023). "'linalg' Dialect." Retrieved June 24, 2024, from https://mlir.llvm.org/docs/Dialects/Linalg/.

LLVM Project (2024). "Multi-Level Intermediate Representation Overview." Retrieved June 24, 2024, from https://mlir.llvm.org/.

NVIDIA Corporation & affiliates (2012). "NVVM IR Specification." Retrieved July 1, 2024, from https://docs.nvidia.com/cuda/nvvm-ir-spec/.

NXP (2024). "eIQ Neutron Neural Processing Unit (NPU)." Retrieved June 24, 2024, from https://www.nxp.com/applications/enabling-technologies/ai-and-machine-learning/eiq-neutron-npu:EIQ-NEUTRON-NPU.

Pelke, R., et al. (2023). Mapping of CNNs on multi-core RRAM-based CIM architectures. VLSI-SoC, IEEE.

Tensorflow (2024). "Tensorflow For Edge & Mobile." Retrieved June 21, 2024, from https://www.tensorflow.org/lite.

Tensorflow (2024). "TFLite Micro." Retrieved June 21, 2024, from https://github.com/tensorflow/tflite-micro.

The IREE Authors (2024). "IREE (Intermediate Representation Execution Environment)." Retrieved June 24, 2024, from https://iree.dev/.

The Linux Foundation (2024). "ExecuTorch: End-to-end solution for enabling on-device inference capabilities across mobile and edge devices." Retrieved June 28, 2024, from https://pytorch.org/executorch-overview.

Thürer, B. (2023). "The Inflation of AI: Is More Always Better?". Retrieved 19.07.2024, 2024, from https://towardsdatascience.com/the-inflation-of-ai-is-more-always-better-8ea1be75e0aa.