MASE: An Efficient Representation for Software-Defined ML Hardware System Exploration

Cheng Zhang, Jianyi Cheng, Zhewen Yu and Yiren Zhao
Department of Eletrical and Eletronic Engineering
Imperial College London
London, UK SW7 2AZ
{cheng.zhang122, jianyi.cheng17, zhewen.yu18, a.zhao}@imperial.ac.uk

Abstract

Machine learning (ML) accelerators have been studied and used extensively to compute ML models with high performance and low power. However, designing such accelerators normally takes a long time and requires significant effort. Unfortunately, the pace of development of ML software models is much faster than the accelerator design cycle. Existing design tools and frameworks can provide quick accelerator prototyping, but only for a limited range of models that can fit into a single hardware device, such as an FPGA. Furthermore, with the emergence of large language models, such as GPT-3, there is an increased need for hardware prototyping of these large models within a many-accelerator system. The design space of a many-accelerator system is often huge, involving both software and hardware optimization. To address this, we propose a novel representation named MASE IR (Machine-learning Accelerator System Exploration Intermediate Representation) that describes data types, software algorithms, and hardware design constraints. We believe MASE IR will open new research opportunities for ML system design.

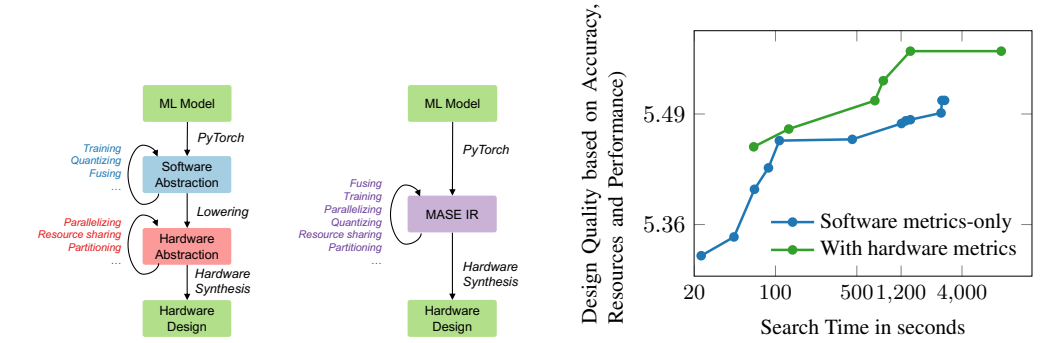
1 Introduction

Designing an efficient ML accelerator requires considerable knowledge and effort from hardware experts. It could take years to implement an ML accelerator in an application-specific integrated circuit (ASIC) and months for prototyping on a reconfigurable device such as a field-programmable gate array (FPGA). The gap between the development cycles of software ML models and hardware accelerators has been widening due to the increasing size of models. Furthermore, new models may significantly alter their program behavior, making the hardware architecture obsolete and necessitating a complete redesign from scratch.

Previous research has explored the co-optimization of software and hardware for Deep Neural Networks (DNNs) from an Automated Machine Learning (AutoML) perspective. The general idea is to include hardware design parameters in the canonical AutoML search space. However, this approach is still relatively limited in terms of hardware architecture exploration. Co-design AutoML typically focuses on a small set of parameters in a Processing Element (PE) array-based architecture [1, 2], such as buffer sizes and PE array dimensions. Moreover, existing Co-design AutoML strategies are constrained to a single type of network, such as Convolutional Neural Networks (CNNs), due to the hindrance posed by the underlying "template" hardware accelerator. The scope of this work is a system-level approach that facilitates mapping various networks to custom silicon.

Many prior DNN hardware accelerator design frameworks [3–6] focus on automating the design of single-device accelerators. This implies that DNNs must fit within the designated device or be reused on the same hardware. The emergence of large language models like GPT [7], which consists of millions or even billions of parameters, has necessitated multi-accelerator systems. This raises the

Machine Learning for Systems Workshop at 37th NeurIPS Conference, 2023, New Orleans, LA, USA.



(a) Traditional Toolflow (b) MASE Toolflow (c) Optimizing `BERT-BASE` in MASE IR

Figure 1: We propose an efficient abstraction named MASE IR that expresses both a software ML model and the hardware architecture of its accelerator. MASE IR opens opportunities for scalable software and hardware co-optimization, leading to improved system quality in terms of accuracy, throughput, and hardware resources.

question of partitioning available hardware resources across multiple devices for different types of DNNs with varying run-time constraints.

In this work, we are interested in the challenge of *a unified abstraction for software and hardware* for ML acceleration. The development flows for ML models and hardware accelerators are currently separate, as illustrated in Figure 1a, where the ML models are treated as a "*read-only*" input to the hardware synthesis tools. What if we lift this restriction by enabling ML model optimization, such as training, in the hardware design flow? What will a unified abstraction look like? How can the abstraction achieve scalability and ensure correctness? We propose a novel representation named MASE IR, which stands for ML Accelerator System Exploration Intermediate Representation, integrated into a hardware architecture exploration tool named MASE to answer these questions. MASE IR enables the concurrent exploration of software and hardware optimizations, as illustrated in Figure 1b. For instance, traditional quantization techniques for general ML models only consider software metrics like accuracy and memory density, while MASE IR enables efficient hardware-aware quantization, considering data parallelism and resource mapping. Figure 1c shows an example of applying hardware-aware quantization on `BERT` in MASE IR, achieving significant improvements in the design quality of the final hardware accelerator system compared to the traditional approach.

2 ML Accelerator System Exploration Intermediate Representation

MASE IR seeks a "trainable" IR that provides a high-level description of ML models and can be transformed back into the PyTorch model. By using the term "trainable", we mean that the optimized IR can be converted back into a PyTorch model that can be retrained via SGD.

MASE IR is a directed graph representation of an ML model and its accelerator architecture, consisting of multiple vertices connected by edges. Each vertex is referred to as a "MASE node". A MASE node represents the computation of an operation at the module level for software (an event) and a custom computation kernel for hardware (an instance). Each edge between two MASE nodes represents data dependence for software and data interface in wires for hardware. MASE IR preserves the original PyTorch module/function within the IR by associating it with the nodes and edges, enabling analysis, transformation, training, and execution of the model.

Figure 2 provides several examples of MASE IR. The leftmost part of the figure shows the initial representation of an input model. The IR consists of six nodes, where the orange nodes represent nodes that do not alter the tensor values, and the blue nodes compute tensors and generate new values. The colors of the nodes serve as annotations, and all the nodes are treated the same by the tool. The node types in MASE IR are a superset of the types supported by PyTorch built-in modules and functions, enabling a direct translation between MASE and PyTorch.

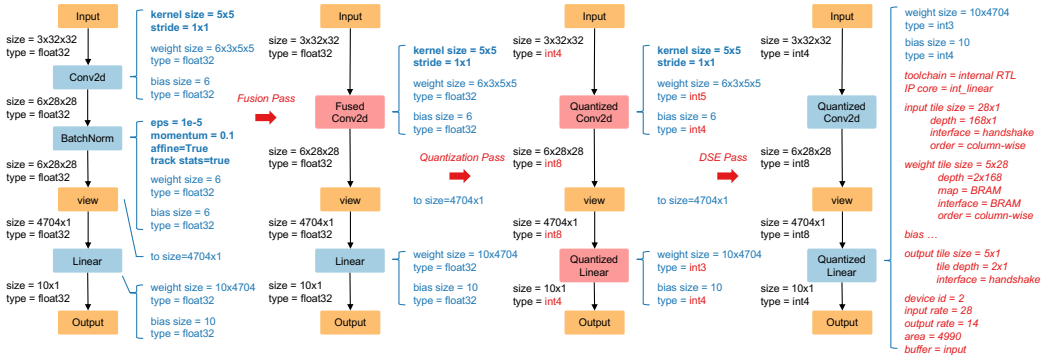


Figure 2: An example of MASE IR being transformed by a sequence of transform passes. The changed parts between two consecutive IRs are highlighted in red. MASE passes can change an ML model in both IR and attributes. These passes can be run out of order because of the same abstraction, which opens up opportunities for software and hardware co-design.

Each instance of a node in MASE IR has a set of attributes that model both software and hardware behavior. In the figure, we show the key attributes for simplicity. The attributes denoted by black text are general and applicable to every node, such as input and output types. Certain attributes (text in blue) are specific to individual node types like the kernel size of Conv2d. The information regarding edges in the IR is captured by the attributes of the preceding and succeeding nodes.

MASE interacts with MASE IR through two forms: actions and passes. An action, also referred to as a pass manager, performs coarse-grained transformations of MASE IR. It consists of a sequence of passes and can be directly utilized by general users ("User interface" in Figure 5). A pass performs finer-grained analysis or transformation. In the following sections, we delve into the process of transforming MASE IR in both software and hardware through a series of MASE passes.

3 Software and Hardware Transformation in MASE IR

The software transformation passes can be classified into two categories: architecture-level and node-level. We present an example of software and hardware transformation in Figure 2. The first pass (fusion pass) conducts software transformation on the model architecture, while the second pass (quantization pass) performs software transformation on the node number format. The third pass is a design space exploration (DSE) pass conducted at both node and architecture level. A detailed explanation is included in supplementary materials. As a software and hardware co-design tool, MASE facilitates the interleaving of software and hardware co-optimization. The passes shown in Figure 2 can be executed in any order, allowing for efficient transformation and creating opportunities for synergistic effects between different passes.

4 Hardware Mapping from MASE IR

The hardware backend of MASE takes a model in MASE IR and translates it into a hardware system. In MASE IR, each MASE node can be mapped into hardware using one of three approaches.

Firstly, MASE has parameterized blocks in its internal hardware library. Each block has a set of inputs and a set of outputs with handshake interfaces. These blocks are manually implemented as templates in RTL or HLS code. If an operation has been defined in the internal hardware library, it could be directly synthesized with custom parameters. These templates have been optimized for high performance and area efficiency. However, the internal hardware library requires manpower to develop and takes a long time to support new operations in state-of-the-art ML models.

The second approach is to import user-defined hardware components. There have been a variety of ML accelerator designs that show high performance and energy efficiency for a particular ML model. These accelerators often contain reusable hardware blocks. MASE enables the integration of external hardware blocks if each data port is implemented using a handshake interface. A user-defined

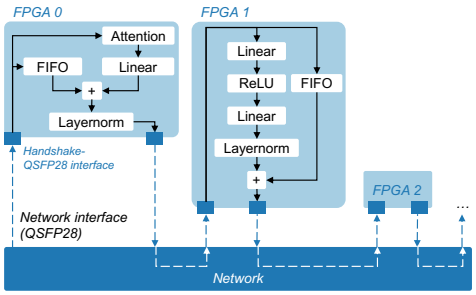


Figure 3: An example of mapping `OPT-125M` onto an FPGA device array in a network. The cross-device communication uses QSFP28 network interface for point-to-point communication. MASE maps the model into a point-to-point device array, which could scale up to hundreds of devices. The device array in the network can grow up at scale with more hardware resources for parallelism.

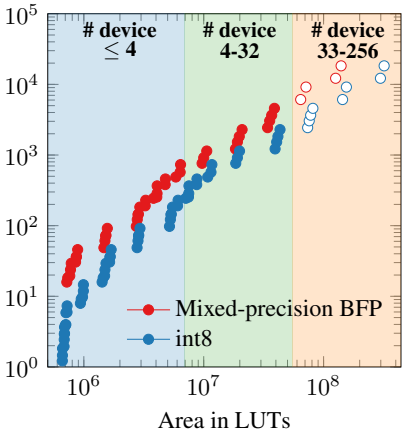


Figure 4: Design points for `OPT-125M` using MASE IR, ranging from a few devices to hundreds of devices. Solid dots represent the tested hardware designs and the hollow dots represent the estimated hardware designs by our simulation model. Compared to the model with int8 types, our hardware-aware quantized models in mixed-precision block floating point (BFP) [8] types could achieve the same performance with fewer resources.

hardware block is treated as a black box. With the parameters of the custom blocks provided, the parallelism of these blocks can be explored with other blocks in the DSE pass.

Thirdly, MASE supports the hardware synthesis of ***arbitrary*** ML models by mapping new operations into hardware using the MLIR-HLS flow. In MASE, each MASE node can be transformed by the Torch-MLIR front end [9] into MLIR, specifically the `linalg` abstraction at the tensor level. It can then be successively transformed into the `affine` abstraction using open-sourced MLIR passes and then translated into HLS code using MASE emit-C pass. These HLS blocks are also connected to other hardware blocks using the handshake interface.

The top-level hardware is a netlist of these hardware blocks at the module level. It is then partitioned and mapped onto multiple devices. Each partition is placed in a wrapper, which translates the handshake interface around the partition to a cross-device protocol, such as QSFP28, for cross-device communication. Figure 3 shows an example of `OPT-125M` being mapped onto multiple U250 FPGA devices. Each device is connected to a shared network using two QSFP28 network interfaces, where each interface has a maximum bandwidth of 100 Gbps. MASE maps the hardware design onto these devices with point-to-point cross-device links. The on-chip kernel communicates with other devices through a handshake-QSFP28 IP core, shown as a small dark blue block in the figure.

5 MASE Tool Flow

MASE is integrated into PyTorch and can directly interface with PyTorch modules without requiring any rewriting. The tool flow depicted in Figure 5 serves as an example of the development flow in MASE. In MASE, all the MASE passes can be applied ***out-of-order***, similar to other compiler frameworks like LLVM [10] and MLIR [11]. The red dashed edges in Figure 5 illustrate the proposed techniques for software and hardware co-optimization. The top right of Figure 5 illustrates an example of how MASE can be used. A detailed explanation is included in supplementary materials. In MASE, an ML model can undergo analysis and transformation in both software and hardware domains. The black edges in Figure 5 depict the fundamental tool flow of MASE.

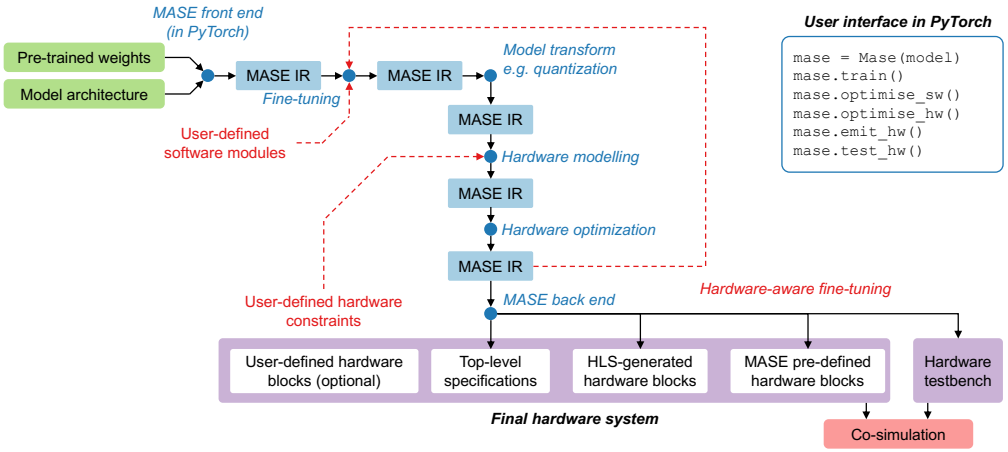


Figure 5: An overview of the MASE tool flow.

6 Conclusion

Models today are evolving rapidly and growing significantly. This makes designing the next ML accelerators for new and large models challenging. We propose an efficient IR called MASE IR that allows the software model and the hardware architecture to be expressed at the same abstraction level. This work represents the initial step of the MASE framework, enabling ML accelerator system exploration within a software compiler flow.

References

[1] Mohamed S Abdelfattah, Łukasz Dudziak, Thomas Chau, Royson Lee, Hyeji Kim, and Nicholas D Lane. Best of both worlds: Automl codesign of a cnn and its hardware accelerator. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[2] Kanghyun Choi, Deokki Hong, Hojae Yoon, Joonsang Yu, Youngsok Kim, and Jinho Lee. Dance: Differentiable accelerator/network co-exploration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 337–342. IEEE, 2021.

[3] Qingcheng Xiao and Yun Liang. Towards agile dnn accelerator design using incremental synthesis on fpgas. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 42–48, 2022.

[4] Michaela Blott, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Trans. Reconfigurable Technol. Syst.*, 11(3), dec 2018. ISSN 1936-7406. doi: 10.1145/3242897. URL `https://doi.org/10.1145/3242897`.

[5] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, Sergo Jindariani, Nhan Tran, Luca P Carloni, Giuseppe Di Guglielmo, Philip Harris, Jeffrey Krupa, et al. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. *arXiv preprint arXiv:2103.05579*, 2021.

[6] Stylianos I. Venieris and Christos-Savvas Bouganis. fpgaconvnet: A framework for mapping convolutional neural networks on fpgas. In *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 40–47, 2016. doi: 10.1109/FCCM.2016.22.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[8] Bita Darvish Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, et al. Pushing the limits of narrow precision inferencing at cloud scale with microsoft floating point. *Advances in neural information processing systems*, 33:10271–10281, 2020.

[9] LLVM. Torch-MLIR, 2023. URL `https://github.com/llvm/torch-mlir`.

[10] Chris Lattner and Vikram Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *International symposium on code generation and optimization, 2004. CGO 2004.*, pages 75–86. IEEE, 2004.

[11] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. Mlir: A compiler infrastructure for the end of moore's law. *arXiv preprint arXiv:2002.11054*, 2020.

Supplementary Material

An example of software/hardware transformation of MASE IR

The fusion pass merges a `BatchNorm` layer following a `Conv2d` layer into the `Conv2d` layer, reducing the hardware area by removing a `BatchNorm` kernel (modifying the model architecture). The quantization pass replaces the `Conv2d` and `Linear` nodes in floating-points with the quantized ones in integer types (modifies the node locally). The forward and backward passes of each node are updated so the node is still trainable. Quantized layers mean efficient computation kernels and lower memory consumption. As shown in the second rightmost IR in Figure 2, the values are quantized from float32 to small integers. Our quantization pass can conduct a mixed-precision search for each parameter in the model to maximize the benefits of quantization at scale.

The hardware transformation can also be conducted at both levels. In Figure 2, we show an example of a design space exploration (DSE) pass. This pass explores the hardware design metrics using the provided specifications, such as available hardware resources and establishes efficient constraints for hardware mapping. There are various design constraints to be explored for each node. For simplicity, we will only highlight the key hardware attributes of a linear layer (text in red):

- The `toolchain` attribute specifies the hardware synthesis approach used, which currently employs pre-defined hardware components provided by MASE.
- MASE offers a range of IP cores, and the `IP core` attribute indicates the specific component to be utilized if a pre-defined hardware component is used.
- Tensors computed by this linear layer are large. To maintain high throughput, matrices are partitioned into tiles and processed in a pipelined manner. The `size` and `depth` attributes specify the tile size and the number of partitions for each parameter, input, and output.
- The `order` attribute schedules the order of tiles fed into the hardware kernels.
- The `interface` attribute determines the hardware interface for each data port, with the current configuration mapping inputs and outputs to handshake interfaces.
- The internal parameters can be stored on-chip or off-chip, depending on the use cases. The `map` attribute specifies how weights are stored for the linear layer.

Overview of MASE tool flow

As we can see in Figure 5,

1. A user-defined model in PyTorch with pre-trained weights is translated into an intermediate representation (IR) in MASE, named MASE IR. The MASE IR of an ML model encompasses both the software algorithm and its corresponding hardware implementation.
2. Similar to most compiler frameworks, a model in MASE IR can undergo software transformations such as post-training quantization using MASE compiler passes.
3. The MASE IR contains comprehensive model information and can be trained or fine-tuned after transformation.
4. MASE IR also describes the accelerator architecture for the model, enabling efficient and scalable hardware transformations using MASE compiler passes, akin to software development.
5. At the backend of MASE, the optimized hardware design can be generated for a scalable accelerator system.
6. MASE incorporates an efficient testing framework to verify the equivalence between the software model and the hardware implementation.

Efficient transformation of new ML models by leveraging the implementations of existing models in both software and hardware. Also, hardware can be further customized in a scalable manner in interactive coordination with software transformations.