PEAX - A Model Augmentation Framework for Adaptive Techniques and Embedded Applications

Max Sponner
max.sponner@infineon.com
Infineon Technologies
Dresden GmbH & Co. KG
Dresden, Germany

Lorenzo Servadei
lorenzo.servadei@tum.com
Chair for Design Automation
Technical University of Munich
Munich, Germany

Bernd Waschneck
bernd.waschneck@infineon.com
Infineon Technologies AG
Neubiberg, Germany

Robert Wille
robert.wille@tum.com
Chair for Design Automation
Technical University of Munich
Munich, Germany

Akash Kumar
akash.kumar@rub.de
Chair of Embedded Systems
Ruhr University Bochum
Bochum, Germany

Abstract

Deploying deep learning models to the edge is challenging due to limited available resources of embedded devices. To address this, we introduce PEAX, a flexible and automatic model augmentation framework designed to enhance deep learning inference efficiency on microcontrollers. PEAX focuses on adaptive and other advanced techniques, including novel approaches to model right-sizing and model slimming, and offers optimizations through graph rewrites and fine-tuning that surpass the capabilities of current deep learning compiler toolchains.

PEAX also implements a compiler interface that converts optimized models directly through TensorFlow Lite for Microcontrollers or microTVM, enabling seamless deployment. We benchmarked its augmentations on an Cortex-M4F-based Microcontroller using both compiler interfaces, demonstrating significant latency improvements of up to 94.2% while maintaining accuracy within 2.9 percentage points of the original model. Our results show that PEAX can substantially improve the performance of deep learning on resource-constrained devices by automatically applying advanced model augmentations, paving the way for more efficient edge AI applications.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; • **Computer systems organization** → *Embedded software.*

Keywords

Adaptive Deep Learning, Embedded Deep Learning, Automatic Configuration

CODAI '25, January 20, 2025, Barcelona, Spain

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Max Sponner, Lorenzo Servadei, Bernd Waschneck, Robert Wille, and Akash Kumar. 2025. PEAX - A Model Augmentation Framework for Adaptive Techniques and Embedded Applications. In *Proceedings of Workshop on Compilers, Deployment, and Tooling for AI (CODAI '25).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The increasing demand for edge AI applications has created a need for efficient deep learning models that can operate on resource-constrained devices such as Microcontrollers (MCUs). However, deploying them on edge devices is challenging due to limited computational resources, memory, and energy constraints. While existing optimization techniques like quantization and pruning can improve efficiency, they are often insufficient on their own to create solutions that are both usable and effective in constrained environments.

To address this challenge, researchers have explored adaptive techniques such as Early Exit Neural Networks (EENNs), which have demonstrated potential for reducing inference costs while maintaining high prediction quality. However, existing approaches often focus on a single technique, creating isolated implementations that need to be maintained individually. Furthermore, current solutions often require significant expertise in deep learning and compiler design, creating a barrier to entry for non-domain experts.

In this paper, we propose PEAX[1], a novel model augmentation framework that bridges this research gap by combining adaptive and static techniques to optimize deep learning models for resource-constrained environments. Our framework's modular design ensures ease of extensibility and minimal conversion costs, making it an attractive solution for deploying deep learning models on edge devices. The key contributions of PEAX include:

- **Performance Improvements:** PEAX applies various augmentations automatically to enhance the performance of deep learning models on embedded devices.
- **Modularity:** Its modular architecture allows easy extension and integration of new techniques.

[1] PEAX [pi:ks] stands for **P**erformance **E**nhancing **A**daptive e**X**ecution

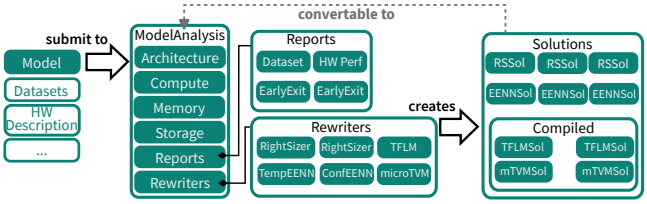

Figure 1: A high-level representation of the overall augmentation flow of PEAX. Multiple `Reports`, `Rewrites` and `Solutions` of the same type can exist within the same `ModelAnalysis`, if they were created with different configurations.

- **Compiler Interface:** PEAX features a compiler interface that facilitates seamless deployment on edge devices through state-of-the-art toolchains.
- **Novel Augmentations:** PEAX introduces innovative approaches to model right-sizing and static slimming, which were developed by leveraging the framework's modular design and reusing existing functionality from early exiting rewrites. These novel augmentations further expand the range of optimization techniques available to users.
- **Comprehensive Reports:** PEAX generates comprehensive HTML-based summaries that provide insight into the submitted models and applied augmentations, lowering the barrier of entry. These summaries are intended to make the performed augmentations explainable to non-domain experts.

For this paper, we implemented the PEAX framework with its modular design, incorporating EENNs augmentation techniques from related work. Additionally, we added two novel static rewrite approaches that enable further model optimizations. We evaluated the effectiveness of these optimizations using targeted compiler flows on an Infineon PSoC 6 MCU, achieving a latency reduction of up to TODO% while maintaining comparable accuracy scores.

Furthermore, we published PEAX on GitHub[2] to encourage further research and development in this area.

2 Related Work

In recent years, adaptive techniques have shown their ability to significantly improved inference costs while maintaining prediction quality [10]. These techniques have also been applied to embedded scenarios and extended by leveraging unique application-specific properties to further enhance performance [5, 6, 8]. Such approaches include the utilization of EENNs with domain-specific at-runtime decision mechanisms, either guided by the input's similarity to the majority class [5], or leveraging the temporal component of sensor data streams [6, 8].

Network Architecture Search (NAS) frameworks for adaptive techniques are able to create efficient models that incorporate adaptive techniques. However, these frameworks rely on costly search methods like evolutionary search and are typically limited to specific problems, such as confidence-based EENNs [2–4]. Furthermore, these frameworks do not always perform all necessary implementation steps, such as configuring decision mechanisms [2–4], and are instead limited to just creating the EENN model architecture or to specific hardware targets or tasks and used layers [4].

In contrast, Network Augmentation (NA) involves starting from an existing model and finding viable solutions at a significantly reduced search cost, proving effective in optimizing models for various applications, which enables the utilization of existing model zoos as starting point [7].

Our work introduces PEAX, a dedicated NA framework designed specifically for embedded deep learning applications. PEAX integrates a range of techniques, including NAs for EENNs and histogram-based termination [7, 8], model right-sizing, and model slimming. Unlike NAS frameworks, PEAX does not rely on costly evolutionary search algorithms and is not limited to a single technique. Instead, it provides a flexible and extensible framework for optimizing deep learning models for embedded applications with a focus on adaptive solutions. This modular architecture enables developers to extend its functionality easily, while reusing existing components during the implementation and execution.

The model slimming and right-sizing augmentations have been created for this publication to showcase the ability to implement novel NAs by reusing the available components already integrated into PEAX.

3 Architecture

The design of PEAX emphasizes three main principles: ease of use, cost-effectiveness, and modularity. These principles ensure that the framework is accessible, cost-efficient in its use, and extensible, facilitating future additions with novel augmentation flows. An overview of PEAX's design is illustrated in Fig. 1.

At the center of the PEAX framework is the `ModelAnalysis` class. This class encapsulates the entire augmentation process for a model, providing a structured approach to optimizing deep learning models for embedded systems. By centralizing the augmentation process into one object, it ensures that each process can access the information available in other components that operate on the same model. This promotes the reuse of information across different optimizations/augmentations and their internal steps, thereby enhancing efficiency and consistency.

3.1 The ModelAnalysis Class

Objects of the `ModelAnalysis` class are instantiated by submitting the original models. They serve as a container for all the information and objects generated during the model's processing. Given that different augmentation flows require vastly different information and processing steps, the `ModelAnalysis` class is structured to include only the base information essential for almost every augmentation that can be quickly acquired. This information includes:

- **Graph-Level Representations:** Both fine-grained and coarse-grained representations of the model are maintained to support various report and rewrite operations.
- **Subgraphs:** The feature extraction and prediction subgraphs in both representations are identified and stored.
- **Cost Estimates:** Detailed node-wise estimates of the model's inference cost in operations per second are calculated.

[2] https://github.com/MaxS1996/peax

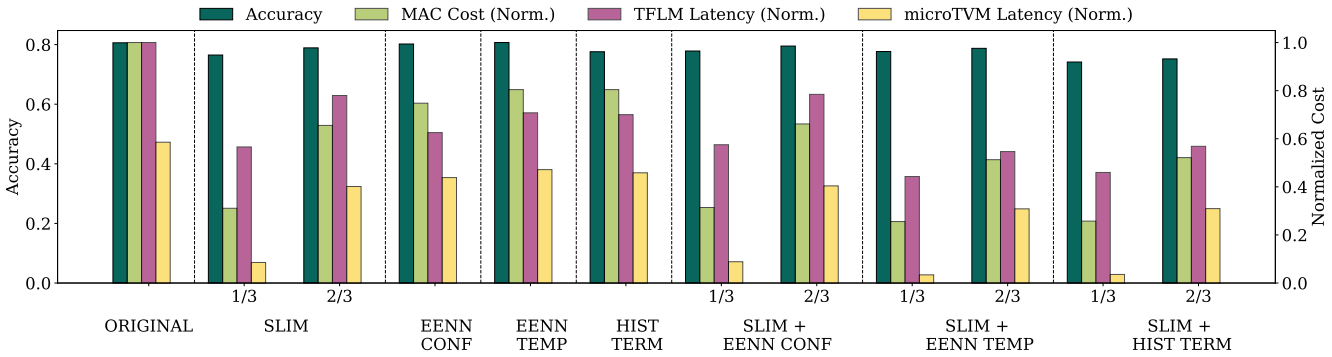

Figure 2: The achieved performances of the different augmentation flows for the MI detection model. The latency values are normalized to the TensorFlow Lite for Microcontrollers (TFLM) latency, to enable a comparison between the supported compiler/deployment toolchains. Detailed results in tabular form will be available in the GitHub repository.

- **Model Task:** The predicted task of the submitted model, such as (binary) classification or regression, is determined and recorded.
- **Data Modality:** The modality of the input data is estimated based on the tensor shape and model architecture.

3.2 Reports

To accommodate the diverse requirements of various augmentation techniques, the `ModelAnalysis` class incorporates a registry of artifacts, known as `Reports`, which provide additional information extracted from the model and can be submitted by the user or `Rewrites` that require their functionality. These `Reports` are evaluated lazily, to ensure efficiency and avoid redundant computations.

Each `Report` is assigned a unique identifier based on its type and configuration, enabling its reuse across multiple augmentations and reducing computational overhead. The `Reports` are submitted to the `ModelAnalysis` as closures that wrap their constructors, and are only created when their functionality is required.

This strategy, including the unique identifier and lazy evaluation, are designed to ensure that the execution of PEAX is cost-effective.

Frequently used `Reports` include `DatasetReports`, which enable developers to submit required datasets for training, evaluation (using the `AccuracyReport`), or finetuning steps as Numpy arrays, Python lists, or TensorFlow Dataset objects. The additional wrapper enables PEAX to extract basic statistics from the datasets, providing developers with insights into the quality of their data, and caching this information for future use.

Other notable `Reports` include the `EarlyExitReport`, which evaluates the submitted model to identify suitable locations for introducing early exit branches and determines the most suitable architecture for each branch location. Additionally, the `HWReport` can be used as an interface to performance models [9], estimating the layer-wise latency, memory, and storage or other performance metric of the model inference. The open-source version of PEAX relies on simple performance models, simplifying the hardware descriptions to the peak operations per second, the available memory and the interconnect speed between processors. Developers can use the `HWReport` class to interface with their own hardware models.

It is important to note, that differently configured `Reports` of the same type can be submitted to the same `ModelAnalysis` object.

3.3 Rewriters

`Rewriters` are a critical component of the PEAX framework, responsible for performing specific augmentations of the model using the base information and `Reports` or other `Rewriters` registered with the `ModelAnalysis` class. These augmentations result in one or more optimized `Solutions`, which can be converted back into `ModelAnalysis` objects, enabling further sequential optimizations.

Multiple `Rewriters` can be registered with the same `ModelAnalysis` to create `Solutions` tailored to different environments or targets. This flexibility allows developers to explore various augmentations and adapt to diverse deployment scenarios, while leveraging the reuse between artifacts.

The workflow within PEAX is managed through a queue system, where `ModelAnalysis` objects have a queue that can be populated with augmentation passes. These passes are closures that wrap the creation of the `Rewriters` and `Solutions` within a single function call. Each `Rewriter` class has a function to enable developers to easily create such a closure to submit to the queue. Each step in the queue produces one or more `Solutions`, which can then be converted into new `ModelAnalysis` objects for subsequent optimization steps in the queue. Alternatively, developers are able to submit `Reports` and `Rewriters` and perform the augmentations manually to have better control over the process. This queue-based system enables efficient and flexible optimization workflows, allowing developers to chain multiple augmentations if needed. In the current version of PEAX, developers still have to select the order and amount of augmentation passes that are added to the queue. These passes can be applied sequentially or in parallel.

One example of a `Rewriter` is the `RightSizingRewriter`, which is a novel static augmentation that was implemented for this publication. This `Rewriter` prunes the submitted model by removing deeper layers and attaching a new classifier subgraph to the deepest remaining layer. Notably, this implementation reuses existing `Reports` from the early exit augmentations of PEAX, including three `DatasetReports` for training, validation, and test sets, an

`EarlyExitReport` to identify potential locations and configurations for the new classifier, and an `AccuracyReport` to compare the options in the search space of potential solutions to the original model. The `AccuracyReport` and `EarlyExitReport` are used to create a search space, where the Pareto front is identified, and potential solutions can be sampled from it. Another novel contribution is the `ModelSlimmingRewriter`, which relies on the `ModelAnalysis` to identify the feature extraction subgraph and create multiple versions of the model by scaling the number of filters and units in the contained layers, while keeping the hyperparameters of the predictor subgraph unchanged. Such a functionality can be used, if a heterogeneous fleet of devices needs to be targeted, without having to handcraft dedicated model architectures for each target.

3.4 Compiler Interface

The PEAX framework features specialized `Rewriters`, known as `CompilerInterfaces`, which convert augmented models into deployable formats using popular toolchains such as TFLM and microTVM. These interfaces enable seamless deployment of optimized models on various target platforms.

One key difference is that `Solutions` generated by `CompilerInterfaces` are not convertible back into `ModelAnalysis` objects, as they are intended for deployment rather than further optimization.

Currently, PEAX supports TFLM and microTVM for model conversion, with microTVM relying on the TFLM interface for the quantization step. The addition of `CompilerInterfaces` and the queueing system enables PEAX to utilize different toolchains for converting and deploying `Solutions` comprising of multiple submodels. Each of these submodels can be converted into its own `ModelAnalysis` object, enabling the usage of the most optimal compiler toolchain for each combination of subgraph and target device.

One promising use case for this capability is the distribution of subgraphs of an EENN across different targets in heterogeneous or distributed environments, allowing for more efficient and flexible deployment of complex models.

3.5 Current Functionality

Several `Rewriters` have been implemented within the PEAX framework:

- **Confidence and Temporal EENNs:** Creates EENNs using confidence-based or temporal decision mechanisms, allowing distribution of subgraphs across heterogeneous devices. Developers can use their own performance models to acquire performance data during the augmentation [7, 8].
- **Histogram-Based Early Termination:** Introduces a pooling layer in a dedicated branch and monitors the histogram of the intermediate feature map (IFM) over time to detect relevant changes in the model input. This rewriter implements an automated augmentation flow for an adaptive technique from related work [8].
- **Model Right-Sizing:** Prunes deeper layers of the model and attaches a new classifier at the deepest remaining layer. This is a static augmentation that reuses implemented `Reports` of the EENN augmentations.
- **Static Slimming:** This augmentation was newly created for this publication. It creates different width versions of the submitted model, reducing training costs by transferring weights between versions. This is useful for creating model variants targeting a heterogeneous fleet of devices.

The static slimming functionality generates multiple model variants with varying numbers of filters or neurons in the feature extraction subgraph. By transferring weights from the original model and previously trained slimmer solutions, the rewriter attempts to minimize the required training effort.

PEAX also offers a mechanism for generating detailed, HTML-based summaries, providing insights into the submitted model and performed augmentations. These summaries enable users to gain a deeper understanding of relevant properties of their model architecture for embedded deployments and the benefits of the augmented solutions.

In conclusion, PEAX's modular architecture, `ModelAnalysis` class, diverse `Rewriters`, specialized `CompilerInterfaces`, and detailed summaries provide a robust and flexible framework for augmenting deep learning models on embedded devices.

4 Evaluation

To demonstrate the capabilities of the PEAX framework, we conducted an evaluation using an Infineon PSoC 6 MCU, specifically its Cortex-M4F CPU running at 150 MHz. We targeted two common embedded deep learning applications: Electrocardiography (ECG) monitoring and speech command detection. These benchmarks were selected because they represent critical and widely-used embedded applications.

We applied all appropriate rewrites implemented in PEAX to the reference models of both use-cases and queued the existing TFLM and microTVM `CompilerInterfaces` to automatically convert the created `Solutions` into a deployable format, this enabled a direct comparison of the performance of both toolchains for these use-cases. Additionally, we evaluated the potential improvements that can be achieved by chaining augmentations for the combination of a static slimming step followed by the application of adaptive techniques.

4.1 Myocardial Infarction Detection on Single-Lead ECG Data

Myocardial infarctions (MIs) pose a significant threat to health, making continuous monitoring crucial for timely interventions. Wearables like smartwatches can be vital in this regard, but they must strike a balance between achieving high accuracy and maintaining long battery life.

For this evaluation, we leveraged a Convolutional Neural Network (CNN) from related work [5], originally designed as an EENN consisting of two 1D-CONV and a dense layer. The original Early Exit (EE) branch was introduced after the first CONV layer. However, for the purpose of this evaluation, we removed the EE branch from the original model to create the baseline model. The evaluation took place on the PTB-XL dataset [11].

The baseline achieved a test-set accuracy of 80.57%, requiring 16,776 Multiply-Accumulate (MAC) operations per inference, resulting in a latency of 9.38 ms (TFLM) or 5.43 ms (microTVM) when

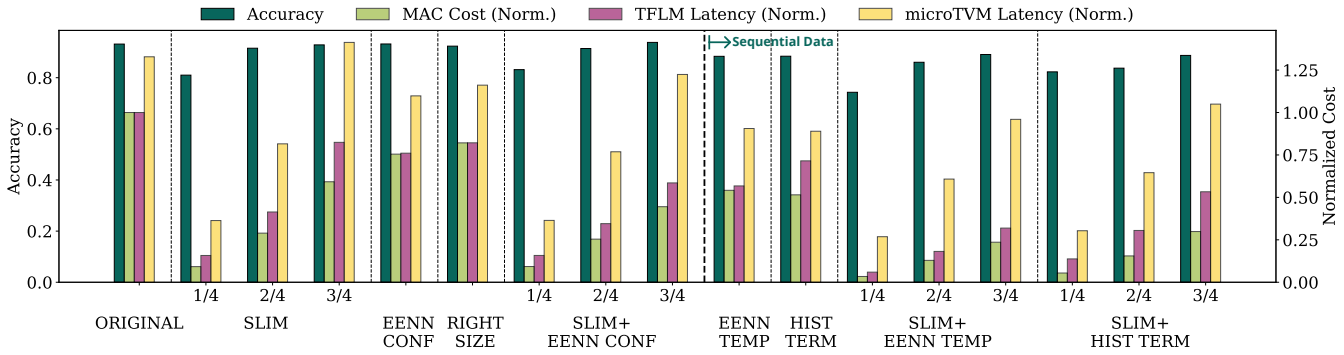

Figure 3: The achieved performances of the different augmentation flows for the mini-speech model. The latencies are normalized to the TFLM latency, to enable a comparison between the supported compiler/deployment toolchains. The temporal EENN and histogram termination solutions have been evaluated on test sequences, which contain temporal correlation between subsequent samples to better represent a real-world environment. Detailed results in tabular form will be available in the GitHub repository.

deployed to the MCU. While the latencies are small enough to be insignificant for the treatment of MIs, reducing them can contribute to longer battery life through race-to-sleep strategies or the deployment to slower, but more energy-efficient MCUs.

Details about the performance of the optimized solutions can be found in Fig. 2. In summary, all solutions reduced the MAC footprint while maintaining competitive accuracy scores.

The smallest slimmed solution reduced MAC operations by 68.9% with an accuracy drop of just 4.1 percentage points (p.p.). Its latency, when deployed via microTVM, was reduced by 85.5%. Compared to a TFLM-based deployment of the original model, the latency could be reduced by 91.45% through model slimming and a microTVM-based deployment.

The ahead-of-time compilation strategy of microTVM generally outperformed TFLM-based solutions, which rely on an at-runtime interpreter, on this use-case.

The ability to chain `Rewriters` was initially designed to directly integrate `CompilerInterfaces` into the augmentation process. However, this feature also enables the combination of various augmentations. To explore the potential of this, we investigated the effects of combining the slimming step with adaptive techniques, including EENNs and histogram-based approaches.

Our experiments revealed that combining slimming with conf.-based EENN did not yield significant performance improvements, likely due to the limited prediction quality and confidence of the EE classifier. In contrast, incorporating temporal decision mechanisms into the slimming process led to substantial reductions in MAC footprint (74.4%) and latency (94.2% when using microTVM, and 55.7% when using TFLM), with a minor accuracy drop of 2.9 p.p.. Combining slimming with histogram-based termination resulted in similar latency reductions, although at the cost of a more significant accuracy loss. Notably, the combination of slimming and confidence-based EENN techniques led to a slight increase in test accuracy on certain configurations, suggesting that there may be additional optimization potential in the automatic configuration step for the training of scaled variants of the model slimming augmentation pass.

Overall, the optimized solutions offer significant improvements in computational efficiency and latency while maintaining high accuracy, making them suitable for real-world healthcare applications.

4.2 Mini-Speech Commands Detection

The task of mini-speech commands detection involves classifying a limited set of speech commands into eleven classes (nine spoken commands, noise, and silence). For this evaluation, we used the DS-CNN Large model from ARM [13] as the submitted model. While smaller configurations of this specific architecture exist, they are not able to achieve the same accuracy levels and it cannot always be assumed that such manually scaled versions are available.

The original model achieved a test-set accuracy of 93.15% and a test-sequence accuracy of 88.70%. The original model had a MAC footprint of 29.3 million MAC operations and latencies of 794.77ms (TFLM), and 1,055.66ms (microTVM).

The test set is the original test split of the mini-speech dataset in its latest version [12], which contains a total of 105,829 audio samples. The test sequences were created from samples of the test set as described in related work [8], totaling thirty minutes of audio containing various speech commands and background noise. This was necessary to create temporally correlated samples similar to real-world scenarios in a reproducible way.

A summary of the performance in terms of maintained accuracy and achieved latency can be found in Fig. 3. The Temporal EENN (EENN TEMP) and histogram-based termination (HIST TERM) solutions utilize the test sequence for evaluation, whereas the other solutions were assessed using the test set.

The most significant reduction in MACs when only applying a single augmentation was achieved through model slimming and temporal solutions (EENN-TEMP and HIST-TERM). The 0.5-width configuration reduced footprint and latency by 28.9% and 41.4% respectively, while maintaining 91.5% accuracy. Temporal solutions reduced MAC and latency by up to 45.8% and 30.6%, but require temporally correlated input data.

TFLM outperformed microTVM in terms of latency. TFLM's interpreter overhead was negligible due to the larger computational footprint and IFM sizes. MicroTVM's deployment was also limited by the insertion of an additional reshape operation in front of the initial convolutional layer, significantly increasing the inference latency.

By combining model slimming with adaptive techniques, specifically the 0.5-width configuration and a temporal EENN, we achieved a significant reduction in mean MAC operations per inference, amounting to 87.1%, while maintaining accuracy levels within 2.7 p.p. of the original model on the sequential data. This, in turn, led to substantial latency reductions, with TFLM experiencing a 81.8% and microTVM a 54% decrease in mean inference latency.

When applied to the original test set, combining slimming with confidence-based EENN techniques, again using the 0.5-width configuration, resulted in a 74.6% reduction in MAC footprint, while maintaining an accuracy score within 1.8 p.p. of the original model. This led to notable latency reductions, with TFLM experiencing a 65.5% decrease and microTVM a 42.9% decrease in mean inference latency.

5 Conclusion and Future Work

The PEAX framework achieves significant performance improvements for embedded deep learning models through dedicated augmentations and optimizations tailored to unique scenarios in embedded use-cases, such as temporally correlated input samples. Existing deep learning compilers lack the backpropagation and optimizing functionality required for such optimizations, but a dedicated framework like PEAX addresses this gap despite having a limited frontend compared to compilers like TVM [1]. The inclusion of a compiler interface allows non-domain experts to use different toolchains without needing specialized deployment pipelines. Our evaluation demonstrated the benefits of targeting various toolchains and deployment methods, as performance varies significantly depending on the toolchain and model architecture.

As demonstrated in the evaluation, PEAX was able to reduced the mean MAC operations by up to 74.4% and latency by up to 94.2% in the MI detection task, while maintaining accuracy within 2.9 p.p. of the original model. In the mini-speech commands task, PEAX reduced MAC operations by up to 87.1% and latency by up to 81.8%, with only a 2.7 p.p. drop in accuracy.

Future work could incorporate a `Rewriter` that automatically selects the best compiler and target device based on performance models created from profiling data and implementing more code generation for the `CompilerInterfaces` to generate a wrapper that unifies the interfaces of the available deployment toolchains. This would further streamline the optimization process and ensure optimal performance for a wide range of embedded deep learning applications. Another potential research direction is to further explore the combination of different augmentations and optimizations, and how the search parameters for the chained optimizations need to be adapted to further improve the inference footprint while maintaining the prediction quality.

Acknowledgments

The project "RadarSkin" has received funding from the German Federal Ministry of Education and Research (BMBF) under the call "Electronic Systems for Edge Computing" (grant number 16ME0543). The responsibility for the content of this publication lies with the author.

References

[1] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Q. Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, Carlos Guestrin, and Arvind Krishnamurthy. 2018. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, 578–594. https://www.usenix.org/conference/osdi18/presentation/chen

[2] Matteo Gambella, Jary Pomponi, Simone Scardapane, and Manuel Roveri. 2024. NACHOS: Neural Architecture Search for Hardware Constrained Early Exit Neural Networks. *CoRR* abs/2401.13330 (2024). https://doi.org/10.48550/ARXIV.2401.13330 arXiv:2401.13330

[3] Matteo Gambella and Manuel Roveri. 2023. EDANAS: Adaptive Neural Architecture Search for Early Exit Neural Networks. In *International Joint Conference on Neural Networks, IJCNN 2023, Gold Coast, Australia, June 18-23, 2023*. IEEE, 1–8. https://doi.org/10.1109/IJCNN54540.2023.10191876

[4] Mohanad Odema, Nafiul Rashid, and Mohammad Abdullah Al Faruque. 2021. EExNAS: Early-Exit Neural Architecture Search Solutions for Low-Power Wearable Devices. In *IEEE/ACM International Symposium on Low Power Electronics and Design, ISLPED 2021, Boston, MA, USA, July 26-28, 2021*. IEEE, 1–6. https://doi.org/10.1109/ISLPED52811.2021.9502503

[5] Nafiul Rashid, Berken Utku Demirel, Mohanad Odema, and Mohammad Abdullah Al Faruque. 2022. Template Matching Based Early Exit CNN for Energy-efficient Myocardial Infarction Detection on Low-power Wearable Devices. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 6, 2 (2022), 68:1–68:22. https://doi.org/10.1145/3534580

[6] Max Sponner, Julius Ott, Lorenzo Servadei, Bernd Waschneck, Robert Wille, and Akash Kumar. 2023. Temporal Patience: Efficient Adaptive Deep Learning for Embedded Radar Data Processing. *CoRR* abs/2309.05686 (2023). https://doi.org/10.48550/ARXIV.2309.05686 arXiv:2309.05686

[7] Max Sponner, Lorenzo Servadei, Bernd Waschneck, Robert Wille, and Akash Kumar. 2024. Efficient Post-Training Augmentation for Adaptive Inference in Heterogeneous and Distributed IoT Environments. In *Proceedings of the 2024 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*.

[8] Max Sponner, Lorenzo Servadei, Bernd Waschneck, Robert Wille, and Akash Kumar. 2024. Harnessing Temporal Information for Efficient Edge AI. In *2024 9th International Conference on Fog and Mobile Edge Computing (FMEC), Malmö, Sweden, September 2-5, 2024*. IEEE, 5–13. https://doi.org/10.1109/FMEC62297.2024.10710223

[9] Max Sponner, Bernd Waschneck, and Akash Kumar. 2022. AI-driven Performance Modeling for AI Inference Workloads. *Electronics* 11, 15 (2022), 2316.

[10] Max Sponner, Bernd Waschneck, and Akash Kumar. 2024. Adapting Neural Networks at Runtime: Current Trends in At-Runtime Optimizations for Deep Learning. *ACM Comput. Surv.* 56, 10 (2024), 248. https://doi.org/10.1145/3657283

[11] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I Lunze, Wojciech Samek, and Tobias Schaeffter. 2020. PTB-XL, a large publicly available electrocardiography dataset. *Scientific data* 7, 1 (2020), 1–15.

[12] Pete Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *CoRR* abs/1804.03209 (2018). arXiv:1804.03209 http://arxiv.org/abs/1804.03209

[13] Yundong Zhang, Naveen Suda, Liangzhen Lai, and Vikas Chandra. 2017. Hello Edge: Keyword Spotting on Microcontrollers. *CoRR* abs/1711.07128 (2017). arXiv:1711.07128 http://arxiv.org/abs/1711.07128