DNN Compiler for Custom AI Chip ISA

CS4099 Project Final Report

Submitted by

Mohammed Ameen	**B210515CS**
Vivek K P	**B210473CS**
Adil Abdul Jabbar	**B210461CS**

Under the Guidance of
Dr. Nirmal Kumar Boran

Department of Computer Science and Engineering
National Institute of Technology Calicut
Calicut, Kerala, India - 673 601

April 15, 2025

NATIONAL INSTITUTE OF TECHNOLOGY
CALICUT, KERALA, INDIA - 673 601

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

तमसो मा ज्योतिर्गमय

2025

CERTIFICATE

Certified that this is a bonafide record of the project work titled

DNN COMPILER FOR CUSTOM AI CHIP ISA

done by

Mohammed Ameen
Vivek K P
Adil Abdul Jabbar

of eighth semester B. Tech in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of the National Institute of Technology Calicut

Project Guide
Dr. Nirmal Kumar Boran
Assistant Professor

Head of Department
Dr. Subashini R
Associate Professor

DECLARATION

We hereby declare that the project titled, **DNN Compiler for Custom AI Chip ISA**, is our own work and that, to the best of our knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degree or diploma of the university or any other institute of higher learning, except where due acknowledgement and reference has been made in the text.

Place : Kattangal
Date : 05/05/2025

Name : Mohammed Ameen Reg. No. : B210515CS	Signature :
Name : Vivek K P Reg. No. : B210473CS	Signature :
Name : Adil Abdul Jabbar Reg. No. : B210461CS	Signature :

Abstract

Our project focuses on developing a DNN compiler that translates deep neural network models from Python-based AI frameworks to a custom AI chip's Instruction Set Architecture (ISA), designed by the **Indian Space Research Organization (ISRO)**. The compiler converts models, built and trained using libraries like TensorFlow and PyTorch, into optimized machine instructions executed by the custom ASIC

The ASIC features a CISC-style ISA, capable of performing neural network operations like convolution and pooling with just a few instructions. The compiler is responsible for extracting layer-wise information, generating instructions for each layer, managing data dependencies, and optimizing memory usage. It also handles weight extraction, preprocessing, and memory arrangement to match the chip's expectations.

This project contributes to ISRO's larger vision of deploying custom ASICs in future space missions for tasks like real-time image processing and AI inference, where such chips outperform conventional GPUs under strict constraints like power limitations and harsh environments.

ACKNOWLEDGEMENT

It is with immense gratitude that we acknowledge the support and guidance received from several individuals, without whom the successful completion of this project would not have been possible.

First and foremost, we would like to express our sincere thanks to **Dr. Nirmal Kumar Boran**, our project guide, for his invaluable encouragement, continuous support, and insightful guidance throughout the course of this work. His expertise and mentorship played a pivotal role in shaping the direction and outcome of our project.

We extend our heartfelt appreciation to **Piyush** and **Kaustubh** from **URSC, ISRO**, key members of the ASIC chip development team, for their constant coordination, technical insights, and unwavering support in integrating our compiler with the hardware development process.

Finally, we thank all faculty members, peers, and family members who have directly or indirectly supported and encouraged us during this project.

Contents

List of Figures

Chapter 1

Introduction

Artificial Intelligence (AI) has emerged as a transformative technology in space exploration, enabling autonomous decision-making for complex tasks such as satellite image analysis, planetary rover navigation, and real-time sensor data processing. Among AI techniques, **Deep Neural Networks (DNNs)** have demonstrated state-of-the-art performance in domains like image classification, object detection, and time-series analysis. DNNs are structured as layered computational graphs—comprising convolutional, pooling, and fully connected layers—that extract hierarchical features from raw inputs via iterative training.

Despite their advantages, the deployment of AI models in space environments is constrained by several critical factors: **power consumption, physical size, weight, and radiation tolerance**. Conventional inference platforms, such as general-purpose GPUs, are unsuitable for space missions due to their high energy demands and limited environmental robustness. These limitations necessitate the development of specialized, low-power hardware accelerators tailored to the rigors of space-based deployment.

To address this challenge, the **Indian Space Research Organization (ISRO)** is developing a custom **Application-Specific Integrated Circuit (ASIC)** designed specifically for DNN inference in space. This

ASIC is engineered to efficiently execute fundamental neural network operations—including convolution, pooling, and dense layer computation—while minimizing power usage. The ASIC leverages a **Complex Instruction Set Computing (CISC)**-style **Instruction Set Architecture (ISA)**, purpose-built to execute DNN workloads using compact and optimized instruction sequences.

1.1 System Overview

The overall inference process is coordinated by a dedicated **master processor**, which is responsible for initializing the ASIC by loading data and parameters into the appropriate **frame and filter memory** regions in RAM. The master processor orchestrates the transfer of input feature maps, kernel weights, and DSP parameters, and then triggers execution on the ASIC via control signals. This architecture ensures energy-efficient inference execution that adheres to the strict reliability and operational constraints of space missions.

This work focuses on the development of the supporting **software toolchain** required to interface with the ASIC, with particular emphasis on the design of a dedicated **DNN compiler**. This compiler serves as the bridge between high-level AI models and the ASIC's custom ISA, enabling efficient, real-time AI inference tailored for ISRO's future spaceborne platforms. The subsequent chapters will explore the system design, compiler architecture, implementation strategy, and performance evaluation in detail.

1.2 Supported Layers

The hardware provides native support for:

- **Convolution layer**

- **Maxpooling layer**
- **Dense (Fully Connected) layer**

Additionally, it implicitly supports:

- **Batch Normalization layer** — integrated into convolution instructions using DSP parameters (see Section **??**).

1.3 Memory Layout

The ASIC is connected to a unified physical RAM that stores all tensors, weights, and parameters. The ISA defines two distinct virtual memory regions:

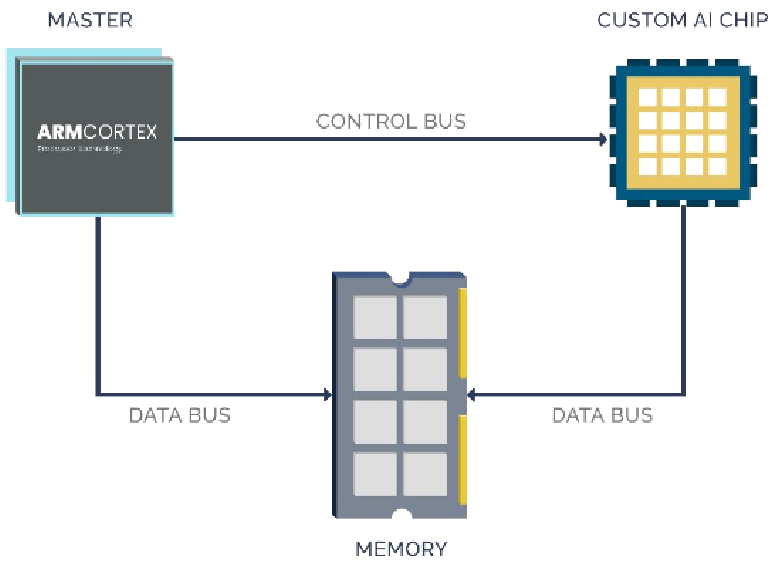

Figure 1.1: Diagram depicting the connection between between master processor and the AI chip.

- **Frame Memory**: stores input data, intermediate activations, and final outputs.
- **Filter Memory**: stores layer weights, DSP parameters, bias vectors and intermediate split results.

Although both memory types reside at arbitrary locations in RAM, the ISA treats each as starting from address 0.

1.3.1 Frame Memory

Frame memory holds input tensors and is updated dynamically during inference with layer-wise outputs. It is initialized using:

1.3.2 Filter Memory

Filter memory is divided into two sections:

- **Weights**: Convolution filters, dense layer weights, and maxpooling placeholders.

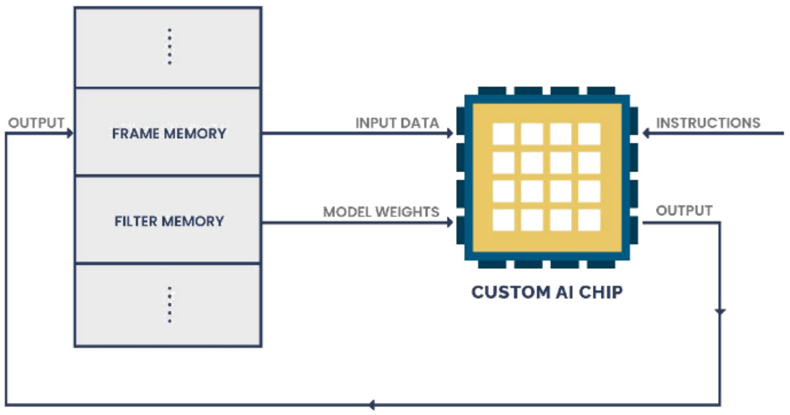

Figure 1.2: Overview of the ASIC AI Chip

- **Parameters**: DSP parameters and bias vectors.

1.3.3 Data Representation

In the ASIC's memory layout, data is organized in a **channel-major** format:

- **For tensors:** `channel` → `height` → `width`
- **For filters:** `filter` → `channel` → `height` → `width`

This means that all data for a given channel is stored *row by row* before moving to the next channel.

1.4 DSP Parameters

The hardware supports a fused operation using three **DSP parameters** per output channel: v_1, v_2, and v_3. This enables bias addition and batch normalization through a single transformation, applied as:

$$y = v_2 + v_1 \cdot (x + v_3)$$

- x: Intermediate value after weighted summation.
- y: Final output after applying DSP transformation.

1.5 Zero Padding

Unlike Python frameworks, the ASIC requires input tensors to be **pre-padded with zeros**. Therefore, zero-padding parameters for a given layer are associated with its **preceding layer**.

During execution, the chip automatically applies the specified padding to the preceding layer's output before it is consumed by the current layer.

1.6 Supported Activation Functions

The hardware supports the following activation functions:

- **Linear** (no activation)
- **ReLU**
- **Leaky ReLU**

ReLU and Leaky ReLU are implemented via a unified operation:

```
if x < v1:
  y = v2 * x
else:
  y = x
```

where $v1$ and $v2$ are configurable parameters.

Chapter 2

Literature Survey

The development of efficient AI compilers for specialized hardware has emerged as a critical research area at the intersection of computer architecture and deep learning. This chapter systematically examines prior work across three key domains relevant to our project, analyzing both the strengths and limitations of existing approaches while identifying the unique contributions of our work in the context of space-grade AI acceleration.

2.1 Hardware Accelerators for Space Applications

Recent advancements in space-grade AI hardware have revealed a clear trajectory toward specialized architectures. NASA's pioneering work on radiation-hardened FPGA implementations [1] demonstrated a 40

2.2 DNN Compiler Architectures

The landscape of DNN compiler architectures presents diverse approaches to bridging high-level models with hardware implementations. The TVM framework [2] introduced a groundbreaking modular stack architecture capable of supporting multiple hardware backends, though its general-purpose design lacks specific optimizations for space applications. MLIR [3] advanced the field through its innovative intermediate representation, offering unprecedented flexibility at the cost of some overhead that may be prohibitive in resource-constrained environments. Perhaps most instructive for our work, Google's EdgeTPU compiler [4] demonstrated the substantial benefits of hardware-aware optimizations, achieving threefold efficiency improvements over generic solutions. These architectural paradigms have significantly informed our compiler design while necessitating novel adaptations to meet the unique demands of space-grade hardware.

2.3 Memory Optimization Techniques

The memory architecture of our target ASIC demands particularly innovative optimization strategies. Recent research on systolic array implementations [5] has shown that meticulous dataflow management can reduce memory bandwidth requirements by up to 60

2.4 Gaps Addressed by Our Work

The comprehensive review of existing literature reveals several critical gaps that our work specifically addresses. Most notably, no prior compiler solution supports the distinctive CISC-style ISA implemented in ISRO's space-grade ASIC, creating a fundamental incompatibility with existing toolchains. Furthermore, current frameworks lack support for the computation splitting paradigm required by the ASIC's limited processing elements, a constraint that demands novel compilation strategies. Our compiler architecture synthesizes the most effective elements from prior work while introducing innovative solutions tailored to space-specific constraints, as will be detailed in the subsequent chapters of this report. This dual approach of leveraging established best practices while developing targeted innovations positions our work to make unique contributions to the field of space-grade AI acceleration.

Chapter 3

Problem Definition

Deep Neural Networks (DNNs) are increasingly used in space missions for tasks like image processing, autonomous navigation, and onboard decision-making. However, conventional GPU-based inference systems are unsuitable for space applications due to high power consumption, size, and lack of environmental resilience.

To address these challenges, the Indian Space Research Organization (ISRO) is developing a custom Application-Specific Integrated Circuit (ASIC) with a CISC-style Instruction Set Architecture (ISA), optimized for AI inference in harsh, resource-constrained environments. While the hardware is being developed, a major gap exists in translating trained AI models into machine instructions compatible with this ASIC.

This project aims to develop a compiler that converts models, built using Python frameworks into optimized machine code for the ASIC. The compiler handles instruction generation, memory management, and model-to-ISA mapping, enabling real-time, power-efficient AI inference on ISRO's space missions.

Chapter 4

Methodology

This section outlines the step-by-step process followed in designing and implementing the DNN compiler for the custom AI chip. The compiler serves as a bridge between Python-based AI frameworks and the custom Instruction Set Architecture (ISA) of the ASIC. The focus here is on how trained AI models and input data are transformed into machine-level instructions and memory layouts compatible with the chip's architecture.

The compilation process is divided into three major stages: **Frame Memory Initialization**, **Filter Memory Initialization**, and **Instruction Generation**. Each of these is described in detail below.

4.1 System Inputs and Outputs

The compiler accepts two primary inputs:

- A trained AI model in `H5` format, created using frameworks like TensorFlow or PyTorch.
- Input data, typically images or sensor data, to be processed by the model.

The outputs produced by the compiler are:

- A **Frame Memory image file**, representing the initial layout of feature maps in memory.
- A **Filter Memory image file**, containing kernel weights, DSP parameters, and other configuration data.
- A sequence of **machine instructions** written in the ASIC's custom ISA, directing the chip to emulate the model.

The **master processor** is responsible for writing these memory images to their corresponding memory regions in RAM and initiating the ASIC execution.

4.2 Frame Memory Initialization

The **frame memory** is a dedicated virtual memory space used to store both input data and intermediate feature maps generated during model inference.

The compiler reads the *input shape* from the trained model and reshapes the input data accordingly. If the first layer requires zero-padding, this

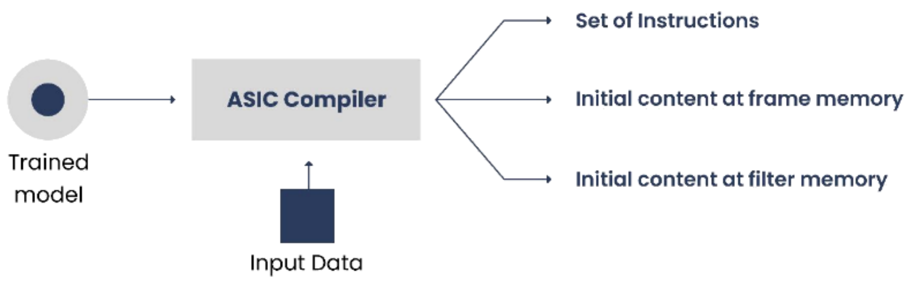
Trained model
ASIC Compiler
Input Data
Set of Instructions
Initial content at frame memory
Initial content at filter memory

Figure 4.1: Overview of the compiler

padding is applied directly to the input before any computation. Unlike typical deep learning frameworks where padding is implicit, the ASIC expects the input tensor to be *pre-padded*, meaning the padding must be present in the input written to memory.

After applying zero-padding, the data is converted into a binary format compatible with the ASIC's memory layout. This binary is written into a file that initializes the frame memory, ensuring the ASIC accesses the input in the correct padded format and memory location at the beginning of inference.

4.3 Filter Memory Initialization

The **filter memory** stores the kernel weights required by convolutional and dense layers, along with the **DSP parameters**, which optimize operations such as bias addition and batch normalization.

The compiler iterates through each relevant layer of the model, performing the following:

- Extracting kernel weights and converting them into binary format.
- Sequentially appending these weights to the filter memory data file.

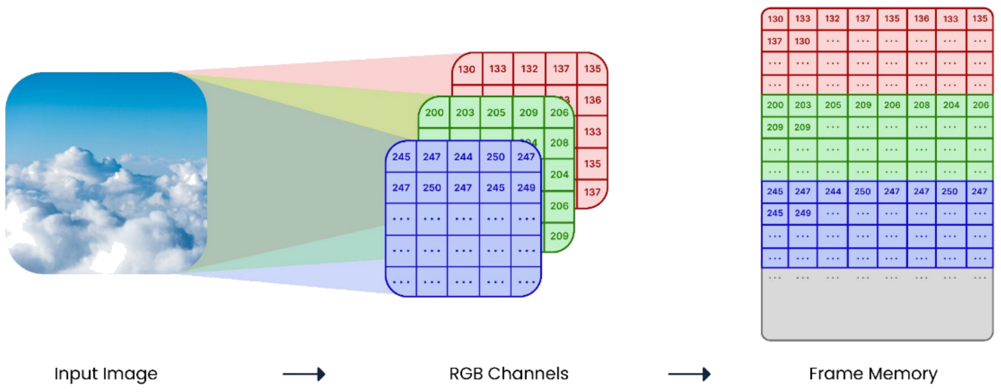

Figure 4.2: Input extraction process for frame memory

- Calculating and appending the corresponding DSP parameters for each operation, based on the model's configuration.

Special handling is required for the first Dense layer after a Flatten layer. Since Keras uses a *height* $\rightarrow$ *width* $\rightarrow$ *channel* memory order and the hardware expects a *channel-major* format, the weights must be **reordered** to align with the hardware's memory layout. This ensures consistency in dot product computations during inference.

At the end of this stage, the filter memory image file contains all the necessary kernel weights and DSP parameters in the correct order for efficient access during inference.

4.3.1 DSP Parameters

The hardware supports a fused **DSP operation** using three parameters per output channel: v_1, v_2, and v_3. This enables both **bias addition** and **batch normalization** in a single step, applied at the *instruction level.*

Without splitting, DSP is integrated into the convolution instruction. **With splitting**, it is applied only in the final addition instruction to ensure correctness after accumulation.

Parameter Computation

- With batch normalization:

$$v_1 = \frac{\gamma}{\sqrt{\text{variance} + \epsilon}}, \quad v_2 = \beta, \quad v_3 = \text{bias} - \text{mean}$$

- Without batch normalization:

$$v_1 = 1, \quad v_2 = 0, \quad v_3 = \text{bias}$$

4.4 Instruction Generation

In this stage, the compiler generates a sequence of machine instructions that direct the ASIC on how to process the model layer by layer. The compiler reads the model sequentially, and for each layer:

- Extracts details such as stride, padding, kernel size, input/output addresses, and DSP parameter addresses.
- Generates instructions that conform to the ASIC's CISC-style ISA, efficiently handling operations like convolution, max pooling, and dense computations.

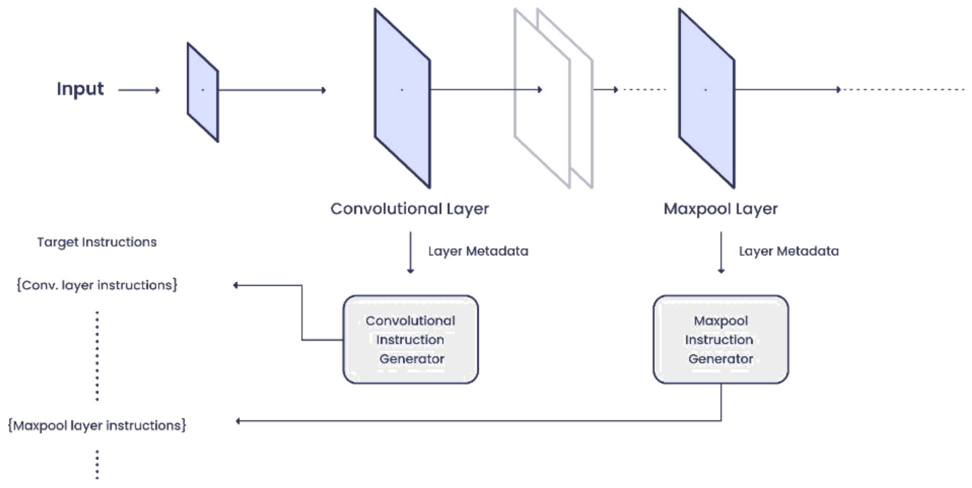
Input
Convolutional Layer
Maxpool Layer
Layer Metadata
Layer Metadata
Target Instructions
{Conv. layer instructions}
{Maxpool layer instructions}
Convolutional Instruction Generator
Maxpool Instruction Generator

Figure 4.3: Instruction generation process

4.4.1 Activation Functions

ReLU and **Leaky ReLU** are implemented using a unified hardware operation:

$$y = \begin{cases} v_2 \cdot x & \text{if } x < v_1 \\ x & \text{otherwise} \end{cases}$$

where x is the input value, y is the activated output, and:

Parameter Calculation

- For ReLU: $v_1 = 0$, $v_2 = 0$
- For Leaky ReLU: $v_1 = 0$, v_2 is the negative slope (e.g., 0.01)

4.5 Hardware Limitations and Splitting

In the AI chip, the **block size** refers to the total number of input tensor elements processed together to compute a single output value. This is equivalent to the size of the convolution filter:

$$\text{Block size} = \text{filter height} \times \text{filter width} \times \text{filter depth}$$

This block is the smallest unit that must be computed in a single cycle. If the available parallel processing elements are fewer than the block size, the compiler automatically **splits** the convolution or dense operation into smaller sub-blocks. These partial computations are independently executed and their results are accumulated to generate the final output.

The same mechanism is applied to **Dense layers**, which are treated as a special case of convolution. However, the splitting strategy differs between convolution and dense layers based on data layout and filter structure.

4.6 Summary

This methodology outlines the complete pipeline from parsing a trained model and input data to generating machine-ready instruction sets and memory images for the ASIC chip. The seamless integration of the splitting mechanism within the instruction generation phase allows the compiler to efficiently handle models even under hardware limitations.

The compiler ensures that frame and filter memories are properly initialized, DSP parameters are applied correctly, and the generated instruction stream executes the model reliably on ISRO's custom AI hardware.

Chapter 5

Results

To validate the correctness of the compiler outputs, multiple levels of verification were performed across memory initialization, instruction generation, and hardware inference results.

5.1 Model Compatibility

The compiler currently supports sequential deep neural network (DNN) architectures composed of convolution, batch normalization, maxpooling, and dense layers. This includes standard models such as **AlexNet**, **ResNet**, and **VGGNet**, which have been successfully compiled and verified on the hardware.

5.2 Memory Verification

For each applicable layer, kernel weights and biases were saved and inspected to ensure they are stored in the correct order and format. This step confirmed that both the **Filter Memory image** and associated metadata conformed to the expected structure defined by the ISA and compiler specifications.

5.3 Instruction Verification

Instruction correctness was verified by storing and reviewing metadata about each layer, including:

- Addresses of weights, biases, and DSP parameters
- Input and output tensor addresses

These were cross-checked against expected values to confirm that the generated instructions correctly reference the initialized memory regions.

5.4 Output Verification

To further validate the correctness of execution, intermediate outputs from each layer were generated using a Python model that mimics the ASIC's behavior. These were then compared against the corresponding outputs from the hardware after each layer. The results consistently matched, confirming that the compiler and chip produced functionally correct outputs across all supported layers.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This project has successfully developed a DNN compiler for ISRO's custom AI ASIC, enabling efficient deployment of neural networks in space-constrained environments. The compiler bridges the critical gap between high-level AI models and the ASIC's specialized architecture by:

- Automatically translating trained models (TensorFlow/PyTorch) into optimized ASIC instructions
- Intelligently managing the ASIC's dual-memory hierarchy (frame and filter memory)
- Implementing computation splitting to work within hardware constraints while preserving model accuracy

Validation tests confirmed the compiler's ability to handle fundamental DNN operations (convolutional, pooling, and dense layers) while maintaining functional equivalence with reference implementations. Although the current implementation shows slower processing speeds compared to GPUs, its superior power efficiency and compact form factor make it ideally suited for space applications where energy conservation is paramount.

6.2 Future Work

To enhance the compiler's capabilities and broaden its applications, several key improvements are planned:

6.2.1 Architectural Expansion

- Support for complex networks (UNet, HRNet) with skip connections
- Implementation of attention mechanisms for transformer-based models

6.2.2 Model Format Support

- Expansion beyond current H5 support to include model formats like pth and ONNX.

6.2.3 Performance Optimization

- Advanced memory allocation strategies for large-scale models

6.2.4 Developer Tools

- Integrated debugging and visualization tools
- Profiling capabilities for power/performance analysis

6.3 Final Remarks

This compiler represents a crucial enabling technology for AI-powered space systems, demonstrating that specialized hardware-software co-design can overcome the challenges of deploying neural networks in resource-constrained environments. As both the compiler and ASIC architecture mature, they will play an increasingly important role in ISRO's autonomous space exploration initiatives.

References

[1] J. Anderson and K. Patel, "Radiation-hardened fpga implementations for autonomous space navigation," *IEEE Aerospace Conference Proceedings*, pp. 1–12, 2020.

[2] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, *et al.*, "Tvm: An automated end-to-end optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation*, pp. 578–594, 2018.

[3] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: A compiler infrastructure for the end of moore's law," *arXiv preprint arXiv:2002.11054*, 2020.

[4] N. Jouppi, D. Yoon, M. Ashcraft, *et al.*, "Compiling for edge tpus: Practical optimizations for on-device ml." Google AI Blog, 2019.

[5] Y. Chen and W. Chen, "Memory-efficient systolic arrays for space-grade neural networks," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 57, no. 2, pp. 1023–1035, 2021.

[6] M. Muller, A. Schmidt, and P. Dubois, "Asic-based vision processors for european space applications," *Journal of Spacecraft and Rockets*, vol. 58, no. 3, pp. 789–801, 2021.

[7] R. Sharma and A. Desai, "Power-efficient cnn accelerator design for indian space missions," Tech. Rep. TR-2022-AI-004, ISRO, 2022.

[8] S. Gupta and L. Li, "Compressed weight encoding for radiation-hardened neural networks," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, pp. 1–10, 2020.

[9] A. Kumar and P. Zhang, "Double-buffering techniques for real-time space applications," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5s, pp. 1–22, 2019.