2025 5th International Conference on Artificial Intelligence and Industrial Technology Applications (AIITA)

Designing and implementing AI acceleration methods based on RISC-V CPU

Xiaoyu Tong
Institute of Data Management and Applications
China Academy of Industrial Internet
Beijing, China
tongxiaoyu@china-aii.com

Jiehao Chen
Institute of Data Management and Applications
China Academy of Industrial Internet
Beijing, China
*Corresponding author: chenjiehao@china-aii.com

***Abstract*—RISC-V, one of the three major instruction sets along with X86 and ARM, provides unrestricted ecological sources with its open-source feature, bringing the opportunity to reshuffle the hardware and software ecosystem. This paper introduces an NPU (neural network processing unit) on RISC-V to build a heterogeneous collaborative computing platform. customized the embedded operating system with Yocto Project; adapted the core software stack; integrated PyTorch into the RISC-V CPU and deployed the optimized ResNet18, YoloV5s, and SSD-VGG16 neural networks to infer offline. The experimental results show that using the method proposed in this paper, NPU show significant advantages in processing neural network inference tasks while maintaining consistent inference results and similar accuracy. For the ResNet18 model, the NPU inference time is 86.792% better than the CPU; for the YoloV5s model, it is 56.27% better than the CPU; and for the SSD-VGG16 model, it is 96.38% better than the CPU. The method implements AI acceleration in the RISC-V architecture. It thus provides a theoretical basis for the ecological interface between NPU and RISC-V.**

Keywords-RISC-V; NPU; Yocto Project; PyTorch

I. INTRODUCTION

The instruction set is a collection of instructions that are used in CPU to perform computations and control the computer system[1], these instructions can be classified into complex instruction set computing (CISC) and reduced instruction set computing (RISC)[2]. RISC-V is an open instruction set architecture (ISA)[3]. Compared to X86 and ARM, in terms of the instruction set, it has a small architectural length, support for scalable instructions and a small number of instructions; in terms of open-source features, it breaks the closed ecology of traditional architectures and has no licensing cost barriers; in terms of power consumption, it supports a variety of low-power modes such as sleep, stop and standby and has become the third largest CPU architecture[4].

Currently, CPU based on RISC-V architecture design are mainly used in the embedded field. There are two categories : the first one is a low-power MCU in IoT; the other is high-performance embedded CPU that can be used for edge computing[5]. The development of deep learning and other computationally intensive artificial intelligence methods is mainly dependent on large power-consuming machines such as server clusters and deep learning graphics cards[6]. Although artificial intelligence (AI) is becoming increasingly prevalent in edge computing devices, the development of low-power AI devices for embedded and Internet of Things (IoT) applications is still in its early stages. And most of the processors responsible for the control work are ARM[7], there is less research into RISC-V based AI acceleration.

This paper proposes an AI acceleration method based on RISC-V CPU to address the above problems. The innovation of the method lies in the co-design of hardware and software. This encompasses the selection of hardware devices, customization of the operating system, adaptation development of the core software stack, and the deployment of model compression techniques on the edge. This includes hardware synergy for AI acceleration: SiFive Unmatched is selected as the primary processor platform for heterogeneous computing, and Cambricon MLU270 serves as the AI acceleration chip; build an embedded operating system for RISC-V CPUs with the custom kernel, boot loader, rootfs, drivers, etc. from the Yocto Project; adapted to develop the core software stack: Computer Vision Library (OpenCV), Cambricon Neuware Runtime Library(CNRT), Cambricon Neuware Driver Interface Library(CNDrv), a deep learning framework PyTorch[8]; ResNet18[9], YoloV5s[10], and SSD-VGG16[11] are used on the edge side using model compression to complete the offline inference of image classification and detection in this hardware and software environment.

II. YOCTO PROJECT BUILD EMBEDDED SYSTEM

A. *Technical background*

Yocto Project supported by the Linux Open Source Foundation, is an open-source collaborative project that provides templates, tools, and methodologies that remove the influence of hardware architecture to create custom Linux-based operating systems for use in embedded hardware platforms and IoT devices.

Yocto Project implementation process[12-16]: capture the upstream software source code, use metadata (recipe) to describe the compilation rules file, use meta-tools (bitbake) to parse, run the build system, generate the kernel, device tree, root filesystem, and SDK. Metadata is the core of a Yocto project and defines a set of semantic information that describes how to build: download source code, apply specific patches, configure dependencies, compile and install, and package the results of the build in a specific format, e.g. deb, rpm, ipk. Metatools can abstract the process of building software: fetching, unpacking, patching, configuring, compiling, packaging, and other tasks; at the same time, it can also rely on the dependencies between different software to perform compilation in an orderly manner.

B. *Building process*

The construction of operating system represents the central objective of embedded Linux project. In accordance with the technical solution proposed in this thesis, the Linux operating system must be tailored to suit specific requirements. The customization process is illustrated in Figure 1 and comprises four principal stages: firstly, assessing the necessary modifications; secondly, configuring the build directory; thirdly, compiling the build; and finally, generating the mirroring.

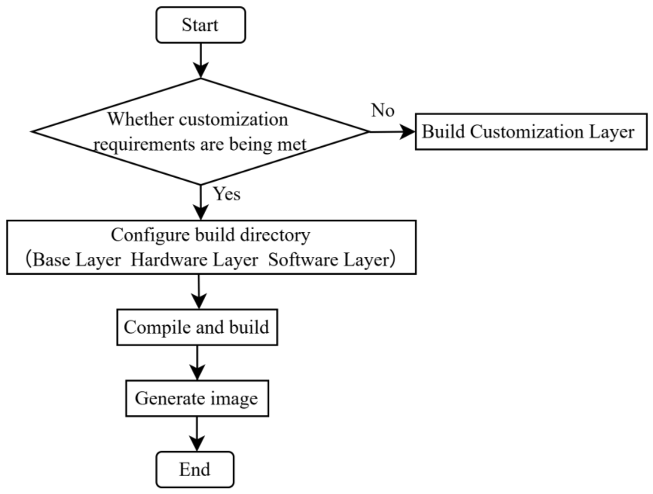
Start
Whether customization requirements are being met
No
Build Customization Layer
Yes
Configure build directory (Base Layer Hardware Layer Software Layer)
Compile and build
Generate image
End

Figure 1. Flow chart of a custom embedded operating system

Determine the requirements: first, select the demo_core_cli.bb file as the mirroring file for the generated system. This recipe file can support the SiFive Unmatched to boot normally. Next, determine whether the information in this recipe file meets the experiment's customization requirements. According to the technical solution, it is known that due to the need to implement the inference function of the Cambricon intelligent accelerator card NPU in the RISC-V CPU, the receipe file needs to be revised according to the requirements.

Configure the build directory: set up the build environment. Using 'source . oe-init-build-env build' to create the build directory, all operations will be done under build. By using the source command, two important files bblayers.conf and local.conf are generated under build, bblayers.conf: build the meta layer used by the embedded OS, this thesis adds a custom base layer, hardware layer, software layer, and customization layer to this file; local.conf: define the environment variables for building the OS. Modify MACHINE variable of the file's target architecture to RISC-V 64; change PACKAGE_CLASSES to specify the package data to be used; set the location where the source code download will be stored and the build attributes, and so on.

Compile and Build: The entire build system builds a complete custom embedded Linux operating system based on the 'bitbake <target mirroring name>' command, which performs thousands of basic tasks, consuming large amounts of disk space and hours. By launching the bitbake engine to build a dependency tree for the execution of the requested work and all the required individual tasks, and then automatically executing them sequentially in the correct order, the dependency tree is characterized not only by describing the mandatory and non-mandatory dependencies that build the recipe of the software components, but also by describing the dependencies and the software version, managing the complexity between the software.

Generate Mirroring: the build directory is compiled by the engine meta-tool (bitbake), which outputs system files related to the build target. Under the build/tmp/deploy/images/unmatched path, will get t kernel, roofs, dtb, and uboot. After successful compilation, the customized system mirroring can be used on embedded devices.

III. Adaptive Development Core Software Stack

The collaborative hardware and software approach to AI acceleration, as outlined in this paper, employs the Cambricon Intelligent Acceleration Card. Although the accelerator card is compatible with CPU architectures such as X86 and ARM for Linux distributions (Ubuntu, CentOS, and Debian), it is currently unable to support RISC-V. Considering this, explore the integration of OpenCV, CNRT, CNDrv, Cambricon Driver, and PyTorch, which are integrated into the meta layer of the Yocto project. For this customized embedded operating system, the necessary software support was provided.

OpenCV, CNRT, CNDrv, Cambricon Driver in meta-cambricon and PyTorch in meta-pytorch. The above meta layer consists mainly of classes, conf, and recipes-*, etc., where classes are class files, all in .bbclass; conf is a configuration file that contains the layer.conf (layer configuration) file. Layer configuration file: add the path of the current layer to the search path of bitbake compilation system, define the paths of all bb files inside current recipes; add a unique identifier to the layer so that the Bitbake can quickly find the meta layer; set the priority of the recipes in this layer and let the build system decide the order in which they are used; specifies the names of other layers that the current meta layer is dependent on; determines the version of Yocto that the current meta layer is compatible with. Recipes-* folders containing *.bbclass (*.bb) and *.bbappend. The *.bbclass contains a license, source code, code version, patch files, environment variables, compilation methods, and installation paths, etc.*.bbappend files extend the functionality of *.bbclass files. Additionally, *.bbclass can inherit common shared functionality from class files.

IV. Model optimization deployment

As the accuracy of the CNN improves[17], its structure becomes increasingly complex, and the number of parameters grows exponentially. However, this level of complexity and a large number of parameters limits its application to devices with limited resources, such as mobile devices, embedded systems, and edge computing platforms. Model compression is a viable way to achieve efficient deep learning on resource-constrained devices.

Parametric quantization is one of the model compression methods that transforms continuous values (or a large number of discrete values) of a signal into a finite number of discrete values to simplify data processing[18]. Construct a mapping relationship between floating-point and fixed-point to minimize the loss of precision while reducing the number of bits in the data expression. Reduces memory consumption and computational complexity by reducing the number of bits in the weights and activation tensor.Offline quantization is a process of parameter quantization that occurs after a model has been trained. During this process, the weights or activation values are converted into a lower-precision data representation. Calibration data is often used to determine the appropriate range or scaling factor for this conversion.This paper adopts the offline quantization method to eliminate redundancy in the neural network and reduce the number of parameters and structural complexity of the model. The specific process is as follows Figure 2:

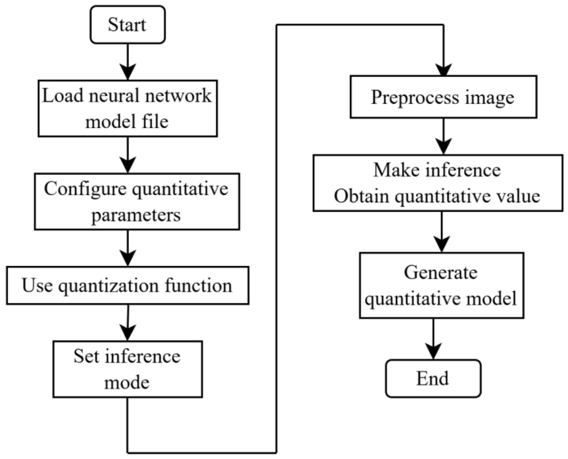
Start
Load neural network model file
Configure quantitative parameters
Use quantization function
Set inference mode
Preprocess image
Make inference Obtain quantitative value
Generate quantitative model
End

Figure 2. Custom Embedded OS Source Directory Structure

V. EXPERIMENTAL RESULTS

A. Building a hardware heterogeneous platform

For the experimental hardware platform designed in this paper, SiFive Unmatched is selected as the main processor, and Cambricon MLU270 is the coprocessor. SiFive Unmatched is powered by the SiFive Freedom U740 (FU740), an SoC that includes a high-performance multi-core, 64-bit dual-issue, with 16GB of DDR4, Gigabit Ethernet, PCIe expansion, USB 3, and M.2 sockets for Wi-Fi, Bluetooth and NVMe storage. This coprocessor is designed for low precision and mixed precision calculations, supporting INT4, INT8, INT16, FP16, and FP32, with 16GB of DDR4. Figure 3 shows the experimental hardware platform:

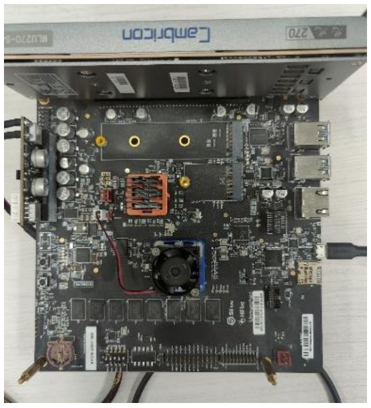

Figure 3. SiFive Unmatched and MLU270

B. Customising embedded operating system

The Yocto Project was used to customize the embedded operating system based on the RISC-V CPU, and the source directory is shown in Figure 4. The layers are as follows: base layer: openembedded-core, bitbake, and meta-openembedded; hardware layer: freedom-u-sdk, meta-sifive and meta-riscv; software layer: meta-clang; and customization layer:meta-pytorch and meta-cambricon.

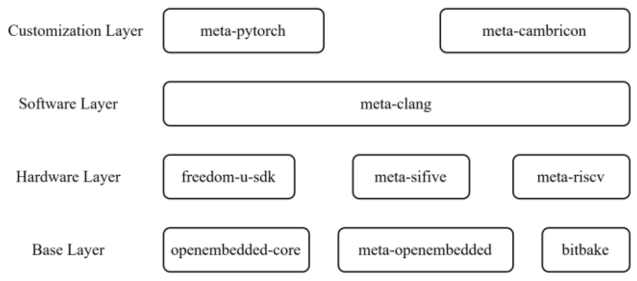

Figure 4. Custom embedded OS source directory structure

Base Layer: from OpenEmbedded project provided and maintained. BitBake is a Yocto Project build system engine, responsible for creating recipes, building the task list, parsing metadata, and executing recipe files. Openembedded-core is a build framework for embedded Linux that provides a cross-compilation solution for compiling Linux distributions of embedded systems, meta-openembedded is an extension of openembedded-core.

Hardware Layer: the Board Support Package (BSP) is customized to the specific embedded hardware device used in the experiment. It includes the necessary configuration, Driver, and bootloader for that hardware. Freedom-u-sdk is an experimental software development kit that provides a complete toolchain suitable for both the QEMU virtual machine and the hardware; meta-sifive is designed for SiFive Freedom E-series and U-series processors; meta-riscv is a generic meta layer offered by SiFive, which can be applied to a range of RISC-V devices.

Software Layer: setting up application packages to enrich operating system functionality. Traditional Yocto Project compilation usually relies on GCC. meta-clang is an alternative to using Clang/LLVM as a GCC compiler and is used in conjunction with other meta layers (e.g. meta-riscv) to provide architecture-specific optimizations and support.

Customization Layer: adding custom content such as specific hardware support, packages, configurations, etc. meta-pytorch defines the process of integrating and using PyTorch in an embedded Linux system, to support running deep learning models on embedded devices. meta-cambricon contains the OpenCV , CNRT , CNDrv and Cambricon Driver. It exists in the form of .bbclass which provides software support for implementing the interface between the NPU and RISC-V.

To customize the embedded operating system, run the following commands in Ubuntu 20.04:

(1) Initialise compilation environment:

```
source ${BUILD_DIR}/freedom_u_sdk/setup.sh
```

(2) Add custom meta-layers to Yocto Project:

```
export BUILD_DIR=/path/to/your/yocto-directory
bitbake-layers add-layer ${BUILD_DIR}/meta-pytorch
bitbake-layers add-layer ${BUILD_DIR}/meta-cambricon
```

(3) Add customization meta layers to mirroring

```
IMAGE_INSTALL_append = "pytorch，cambricon"
```

(4) Set hardware board and execute operation commands

```
MACHINE=unmatched bitbake demo-coreip-cli
```

(5) Get system mirroring file

```
${BUILD_DIR}/deploy/images/unmatched/demo-coreip-cli-unmatched.rootfs.wic.xz
```

C. Evaluate offline models inferring

To verify that AI acceleration can be correctly applied to RISC-V using the proposed implementation. This experiment introduces a control group experiment. The model files for ResNet18, YOLOv5s, and SSD-VGG16, which were developed in PyTorch, are converted to file types compatible with the Cambricon Inference Accelerator Card using the model compression method described in the paper, and then deployed on the custom RISC-V operating system. When performing model inference offline, select the device type as CPU or NPU. The results of the inference are as follows:

(a)ResNet18

(b)YoloV5s

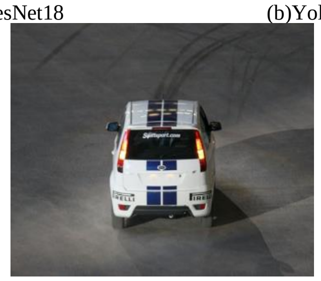
(c)SSD-VGG16

Figure 5. Test data used in inference experiments

TABLE I. COMPARATIVE EXPERIMENTS FOR MODEL INFERENCE

	Accuracy		Time(s)		Inference	
Neural Network	CPU	NPU	CPU	NPU	CPU	NPU
ResNet18	0.837	0.812	76.467	10.100	parachute	parachute
YoloV5s	0.910	0.887	110.368	48.265	bird	bird
SSD-VGG16	1.000	1.000	693.467	25.133	car	car

The inference image used in the experiment is shown in Figure 5. The data in the Table 1 are from results obtained when the hardware platform is rebooted, and the inference is run for the first time, removing the effect of the system cache, hence the high time values.The experimental data shows that the NPU has a significant advantage in processing neural network inference tasks while maintaining consistent inference results and similar accuracy. For the ResNet18 model, the inference time of the NPU is 86.792% higher than that of the CPU; for the YoloV5s model, the inference time of the NPU is 56.27% higher than that of the CPU; and for the SSD-VGG16 model, the inference time of the NPU is 96.38% higher than that of the CPU, showing that the method proposed in this paper has certain feasibility and accuracy in practical applications.

VI. CONCLUSION

This paper focus on the design and implementation of AI acceleration techniques for RISC-V CPU architectures to establish a strong technical foundation for intelligent computing applications. The research covers various aspects such as hardware selection, operating system customization, and the design and integration of AI algorithms. A comprehensive analysis of the RISC-V architecture characteristics and the choice of an appropriate hardware development platform completes the hardware selection. At the same time, the operating system is customized and developed according to the experimental objectives. In terms of construction and integration of AI algorithms, classical neural network algorithms for image classification and target detection are introduced and deployed on the NPU edge using model compression techniques, and specific experiments are performed to validate the feasibility and validity of the methods proposed in this thesis on the RISC-V architecture. In terms of construction and integration of AI algorithms, classical neural network algorithms for image classification and target detection are introduced and deployed on the NPU edge using model compression techniques, and specific experiments are performed to validate the feasibility and validity of the methods proposed in this thesis on the RISC-V architecture.

REFERENCES

[1] Pedersen, J.E., Abreu, S., Jobst, M. et al. Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing. Nat Commun 15, 8122 (2024).

[2] Feng H, Guo J, Hwang J, et al. The relationship between autonomy support from parents and exercise adherence among male Korean adolescents: the multiple mediating effects of resilience and coping style[J]. Journal of Men's Health, 2024, 20(2): 38-50.

[3] Schlägl M, Stockinger M, Große D. A RISC-V “V” VP: Unlocking Vector Processing for Evaluation at the System Level[C]//2024 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2024: 1-6.

[4] Nurmi A, Lindgren P, Kalache A, et al. Atalanta: Open-Source RISC-V Microcontroller for Rust-Based Hard Real-Time Systems[C]// International Conference on Architecture of Computing Systems. Cham: Springer Nature Switzerland, 2024: 316-330.

[5] Chen X H, Hu S P, Liu H C, et al. Research on LLM Acceleration Using the High-Performance RISC-V Processor" Xiangshan"(Nanhu Version) Based on the Open-Source Matrix Instruction Set Extension (Vector Dot Product)[J]. arXiv preprint arXiv:2409.00661, 2024.

[6] Ma J. Extraction and Reconstruction of Traditional Art Visual Elements by Graphic Design Incorporating Deep Learning[J]. Applied

Mathematics and Nonlinear Sciences, 2024, 9(1). DOI:10.2478/amns-2024-2450.

[7] Asghari A, Azgomi H, Barzegarinezhad Z A .Energy-aware server placement in mobile edge computing using trees social relations optimization algorithm[J].Journal of Supercomputing, 2024, 80(5). DOI:10.1007/s11227-023-05692-4.

[8] Paszke A, Gross S, Massa F, et al. Pytorch: An imperative style, high-performance deep learning library[J]. Advances in neural information processing systems, 2019, 32.

[9] He K, Zhang X, Ren S, et al. Identity mappings in deep residual networks[C]//Computer Vision-ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV 14. Springer International Publishing, 2016: 630-645.

[10] Redmon J. You only look once: Unified, real-time object detection [C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[11] Liu W, Anguelov D, Erhan D, et al. Ssd: Single shot multibox detector[C]//Computer Vision-ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part I 14. Springer International Publishing, 2016: 21-37.

[12] Linux Foundation. Yocto Project Quick Build.[EB/OL]. [2024-11]. https://docs.yoctoproject.org/.

[13] Salvador O, Angolini D. Embedded Linux Development with Yocto Project[M]. Packt Publishing Ltd, 2014.

[14] Gonz á lez A. Embedded Linux Projects Using Yocto Project Cookbook[M]. Packt Publishing Ltd, 2015.

[15] Toivanen M. OpenDataPlane (ODP) as a Part of a Linux Operating System Image Built with Yocto Project[J]. 2022.

[16] Demeter L. Graphical User Interface development using Embedded Wizard and Yocto Project for an IVD device[J]. 2023.

[17] Kattenborn T, Leitloff J, Schiefer F, et al. Review on Convolutional Neural Networks (CNN) in vegetation remote sensing[J]. ISPRS journal of photogrammetry and remote sensing, 2021, 173: 24-49.

[18] Misra D, Chaudhary M, Goyal A, et al. Uncovering the Hidden Cost of Model Compression[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2024: 1611-1621.