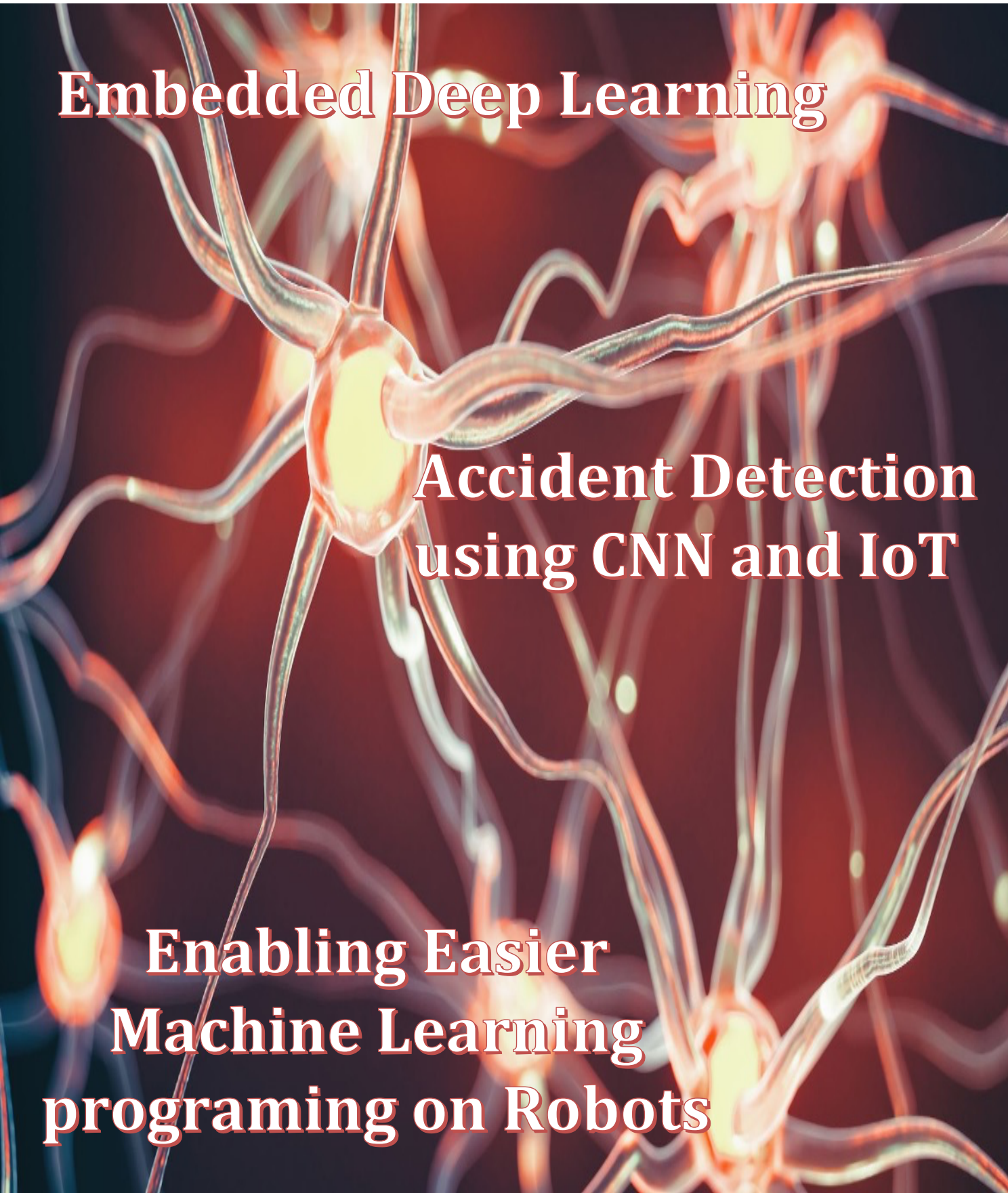
FEEDFORWARD

The Flagship Publication of IEEE COMPUTER SOCIETY

Santa Clara Valley Chapter

olume 1, Issue 1 | From the heart of the Silicon Valley | Jan – Mar 2022



Embedded Deep Learning

Accident Detection using CNN and IoT

Enabling Easier Machine Learning programing on Robots

Website & Media

- https://r6.ieee.org/scv-cs/
- https://www.linkedin.com/company/78437763/
- https://www.linkedin.com/groups/2606895/
- https://www.facebook.com/IEEEComputerSocSCVchapter
- https://twitter.com/IEEEComputerSoc

Mailing List

http://listserv.ieee.org/cgi-bin/wa?SUBED1=cs-chap-scv&A=1

From the Editor's Desk

Dear Readers,

Welcome to the inaugural issue of **Feedforward,** the flagship publication of the IEEE Computer Society, Santa Clara Valley chapter.

Ours is the largest chapter in the Silicon Valley Section of IEEE with over 4,000 subscribers in our mailing list, 1,400+ paid members, and a strong following of over 12,400 on Twitter alone. We are uniquely positioned by the virtue of our location and available expertise. Silicon Valley is a trendsetter. It is not an exaggeration to say that Silicon Valley is at the heart of the innovation that is driving the world economy and the civilization in general. The chapter can
play a vital role in all of this. I started Feedforward with this vision in mind and hope to evolve it into a full-fledged professional trade journal that will be indexed in significant databases.

Along the lines of the vision, I also proposed to start an annual International Conference on Applied Data Science (ICADS) to focus on quick summaries of leading research and experience in the area from all over the world. I am fortunate to be supported by the ExCom that you voted for in all these endeavours. Details about ICADS and other upcoming events can be found inside this issue. True to its international vision, this issue of Feedforward presents articles written by authors from varied geographical regions on cutting-edge topics. Articles for future issues are welcome on all topics related to the IEEE Computer Society charter and if possible, relate to the Silicon Valley. Please submit your articles on our website here: https://r6.ieee.org/scv- cs/?p=2036.

We organized and cosponsored several events this year already and more are in the pipeline. The list of the upcoming events curated during the month of March is posted on our website at https://r6.ieee.org/scv-cs/?page_id=2030.
Please subscribe to the chapter mailing list and follow the chapter social media pages and groups to get timely updates and take advantage of the events. Since most events are online, we often get professionals from as far as Thailand attend our events.

Networking is still one of the primary reasons we all join professional societies. Keeping this objective in mind, the chapter open-house is planned to be in- person in addition to a live broadcast over YouTube and Zoom. Most of the past events of our chapter are available on IEEE.tv https://ieeetv.ieee.org/search?search_q=scv-cs and on YouTube, where they are live broadcasted: https://www.youtube.com/playlist?list=PLLsxQYv4DdJlYcGPwqUJsnHmfqMtB3eSJ

We are looking for more volunteers to help in various roles. You can help as a reviewer of the articles, papers, be a guest editor for special issues, help organize conferences and events, help with the publicity for our events, and more. Please consider being part of the success story by signing up here: https://r6.ieee.org/scv- cs/?p=2039. With the onset of the Spring season, let's Feedforward the chapter to a bright new future. Hope you are all with me in my efforts. Happy reading and happiness always!

With every best wish,
Vishnu S. Pendyala

Monday, March 28, 2022
San Jose, California, USA

Enabling Easier Programming of Machine Learning Algorithms on Robots with oneAPI Toolkits

Denisa Constantinescu, Angeles Navarro, Rafael Asenjo
Computer Architecture Dpt., University of Málaga

Juan-Antonio Fernández-Madrigal, Ana Cruz-Martín
System Engineering and Automation Dpt., University of Málaga

***Abstract*—This work shows that it is feasible to solve large-scale decision-making problems for robot navigation in real-time onboard low-power heterogeneous CPU+iGPU platforms. We can achieve both performance and productivity by carefully selecting the scheduling strategy and programming model. In particular, we remark that the oneAPI programming model creates new opportunities to improve productivity, performance, and efficiency in low-power systems. Our experimental results show that the implementations based on the oneAPI programming model are up to 5× easier to program than those based on OpenCL while incurring only 3 to 8% overhead for low-power systems.**

■ **WHEN** you think of oneAPI use cases that leverage the performance of heterogeneous computing, you might first envision powerful workstations and multi-megawatt supercomputers loaded with big CPUs, GPUs, and FPGAs training complex neural networks. However, our team is working on the opposite end of the spectre. This work explores solving problems in systems that use compute-intensive reinforcement learning (RL) algorithms optimized for battery-powered heterogeneous computing devices.

We have focused for many years on productively exploiting heterogeneous chips leveraging Intel® Threading Building Blocks (TBB) as the orchestrating framework and developing heterogeneous scheduling strategies [7], [8]. So, the announcement of oneAPI was immediately received in our group as an enticing opportunity to raise the level of abstraction in our implementations of heterogeneous schedulers. We see oneAPI as a solid endorsement to SYCL and modern C++ as the basis for the 'homogeneous programming of heterogeneous platforms' idea.

Heterogeneous computing with multiple types of processors can benefit large-scale and supercomputers to embedded systems. However, with smaller systems (as mobile robots), compute-intensive, automated decision-making algorithms need to be efficient, and schedulers need to be aware of energy consumption as they assign tasks to different processor architectures to optimize throughput. These aspects require new approaches to optimizing solutions for low-power systems, which we explore in this work.

Many automated planning and decision-making algorithms rely on Markov Decision Processes (MDPs) [1] and Partially Observable MDPs (POMDPs) [2]. MDPs and POMDPs describe how an intelligent agent with a defined goal learns to make better decisions by doing, even without knowing the map of its environment. POMDP and MDP agents can cope with uncertainty, such as not knowing what lies ahead or whether their actions will be beneficial. The literature shows that solving real-world problems for this kind of agent is not considered for low-power platforms [3] and computing an optimal solution for medium to large-

sized problems is intractable [4]. But the medium-large scale problems include most of the practical applications in autonomous robots, deep-space navigation, search and rescue, inspection and repair, toxic-waste cleanup, and much more.

Consequently, we focus our efforts on optimizing decision-making and planning algorithms for mobile robots (illustrated as the red boxes in **Figure 1**). This niche would highly benefit from a solution for runtime and energy-efficient implementation.

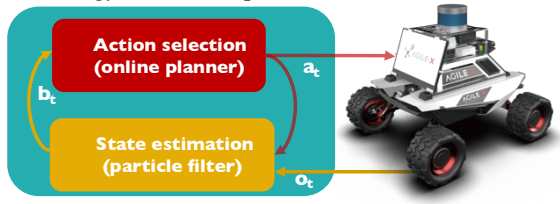


Figure 1. Decision making under uncertainty to plan for the navigation of a mobile robot modelled as a POMDP agent.

The primary source of uncertainty in a POMDP is modeled by maintaining an information state called belief, $b \in B$, representing a probability distribution over the state space S. In practice, a particle filter (the yellow block in **Figure 1**) is used to implement a recursive Bayesian filter to estimate the underlying state s_t of a dynamical system like this: $b_t = b(s_t) \propto p(o_t|s_t) \int p(s_t|s_{t-1}, a_{t-1}) b(s_{t-1}) ds_{t-1}$ [5]. Here, o_t is a sensor measurement and a_t is the action or control command; $p(o_t|s_t)$ models the observations (from sensors) while $p(s_t|s_{t-1}, a_{t-1})$ describes the system's dynamics (e.g., state transition due to actuators).

At any time t, an MDP agent is in one of the possible states, s_t, which is known, while a POMDP agent only knows its belief state estimate b_t. Given either the current state or the belief state, the agent must decide for and execute the action a_t, given a finite set of possible actions. The chosen action is known as the *policy*. Taking action a_t results in an immediate reward $r_t = r(s_t, a_t) \in \mathbb{R}$, and in a transition to state s_{t+1} with probability $p(s_{t+1}|s_t, a_t)$. After each transition, the agent observes o_t, and the process repeats until the agent reaches its goal.

We aim to enable easy and feasible ways to implement decision making on low-power mobile platforms and focus on online planning under uncertainty for practical applications, such as autonomous driving and service robotics, that must run on SoC mobile platforms. These applications often have real-time execution constraints and run on battery-powered platforms.

The main challenge is to keep the runtime and energy performance in check while allowing the users (programmers) to code solvers for decision-making problems. Our proposed solution uses low-power aware heterogeneous computing strategies, sparse data structures to fit real-world size decision-making problems on SoCs (System on Chip) with scarce memory and computing resources, and oneAPI with DPC++ programming [6].

DEVELOPING THE SOLUTION—NAVIGATING NEW SPACES FOR MDP AGENTS

Our team has created new methods, memory-efficient data structures, and low-power aware heterogeneous computing schedulers [3], [7] to enable an intelligent agent to act autonomously in environments where the effects of its actions are not deterministic (**Figure 2**). For example, a rover taking samples from the surface of Mars may not know if a sample is worth taking or if the direction of travel will lead to worthy specimens. Or, a drone looking for survivors trapped after a natural disaster may not know if it is following a path that will eventually lead to a person.

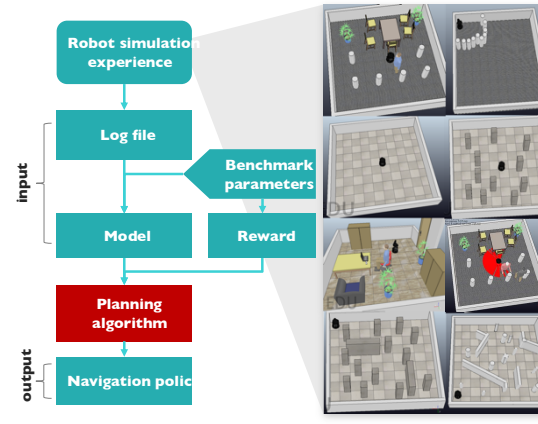


Figure 2. Robots must learn to navigate in uncertain environments, such as those above, using RL methods. We illustrate the configuration of different navigation scenarios in V-REP simulator (the black object is the robot) and how we use the simulated experience to plan navigation policies in new scenarios.

We start our journey with the Value Iteration (VI) algorithm, commonly used in MDPs and a core kernel in many RL methods, optimizing its data structures for memory use and access. Then, we explore ways to improve the performance of this planning and decision-making algorithm. In the flow diagram from **Figure 2**, the VI algorithm is represented by the red block (planning algorithm).

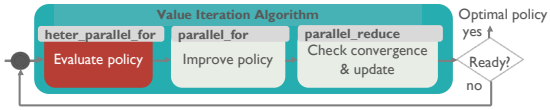


Figure 3. Planning flow with a VI algorithm.

We explore different parallel implementation strategies on low-power SoCs to improve the VI runtime and energy use. We have initially implemented multicore parallelism, using OpenMP and TBB [8], and then included GPU accelerators programmed with OpenCL. Finally, we have added heterogeneous scheduling to balance and optimize the computing resource utilization to minimize runtime and energy consumption on the most resource-consuming kernel of VI (Evaluate policy, in red), as shown in **Figure 3**.
We approach the heterogeneous scheduler code development using three different programming models: OpenCL (OCL), oneAPI with SYCL-style buffers (BUFF) written in DPC++, and oneAPI with unified shared memory (USM) written in DPC++.

EXPERIMENTAL SETUP

For the implementation, testing, and evaluation, we use Intel® DevCloud and two local mobile CPU+iGPU SoCs:

- Kaby Lake - A 1.6 GHz Intel® quad-core CPU i5-8250U, featuring a UHD 620 integrated GPU @300 MHz, 8GB of DDR4 and a TDP of 10 to 15 Watts. OS: Ubuntu 18.04.
- Tiger Lake - A 2.8 GHz Intel® quad-core CPU i7-1165G7, featuring an Iris Xe integrated GPU @1.3 GHz, 16GB of LPDDR4x and a TDP of 12 to 28 Watts. OS: Ubuntu 20.04.

Our parallel implementations use OpenCL, TBB, the oneAPI programming model, the Intel® oneAPI Base Toolkit, and Intel® oneDPL. We have installed the oneAPI DPC++ 2021.1 Compiler and the corresponding Intel® NEO Graphics Drivers on each platform. We measure energy use with PCM library [9].

We have chosen Intel low-to-medium-power processors as testbeds in our experiments because they are energy-efficient and powerful enough to run at least some AI benchmarks onboard a mobile robot. Besides, the quality and ease of use of the Intel profiling, debugging, and supporting tools now included in the Intel® oneAPI toolkits—Intel® VTune™ Profiler, Intel® Advisor and its Flow Graph Analyzer feature, Intel® Inspector—add to the "productivity" factor that we consider key to democratizing parallel and heterogeneous programming.

EVALUATING THREE PROGRAMMING MODELS

The oneAPI programming model creates new opportunities to improve performance and efficiency in low-power systems. We implement three heterogeneous schedulers for orchestrating CPU+GPU execution and evaluate them for low-power use cases.

We use the Cyclomatic complexity (CC) and Programming Effort (PE) metrics to measure how easy (or difficult) it is to program a code [10] and show the results in **Figure 4**. Higher values for CC and PE mean it is more complicated for a programmer to code the algorithm.

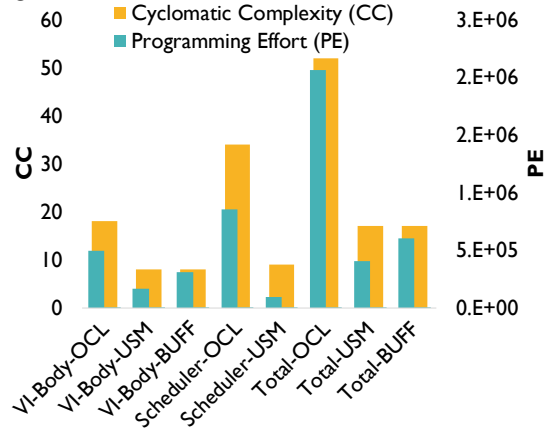


Figure 4. CC and PE results across multiple implementations of heterogeneous schedulers using OpenCL (OCL) and oneAPI (BUFF, USM).

We have found that because DPC++ code is more compact and efficient to program than OpenCL, it is as much as five times easier to program than OpenCL. With a careful scheduling strategy, it only adds three to eight per cent overhead.

Figure 5 shows results with MDP problems using three heterogeneous schedulers for CPU+GPU execution: HO uses a static partitioning strategy, HD a dynamic partitioning strategy, and HL an adaptive strategy. Each scheduler has an OpenCL (OCL) and oneAPI (ONE) version. TBB and OCL are CPU-only and GPU-only implementations, respectively, and the rest are CPU+GPU optimizations.

To configure the best CPU+iGPU workload partition for HO-OCL, we apply brute force offline exploration and tune HO with problem-specific know-how and optimizations. As a result, HO-OCL has no overhead, and the performance improvement is highest, but more painful and time-consuming to get it right. However, applications using the oneAPI Implementation of HL are extremely easy to code in comparison. We pay for productivity with some performance loss compared to HO-OCL due to the abstraction overhead. HD scheduler uses a strategy in

between HO and HL in terms of efficiency and ease of programming.

Figure 5. Speedup and energy improvement results across multiple implementations of heterogeneous schedulers using OpenCL (OCL) and oneAPI (ONE) on the Kaby Lake platform.

From the three scheduling strategies evaluated, static scheduling (HO) performs best in performance and energy efficiency, but it requires exhaustive offline searching. Adaptive scheduling provides good results with no previous training (HL), when using the USM approach to code the kernels and scheduler for large problem sizes.

ENHANCING ONLINE PLANNING FOR POMDP

Next, we apply these lessons learned from optimizing MDP planning with VI [12] to more complex decision-making procedures for POMDP agents.

We enhance a state-of-the-art POMDP online planner, DESPOT-α [11], by adding an efficient experience memory based on Bloom filters to it. This data structure is used to recall policies from experience with similarity search. In our preliminary evaluation, we compare the planning time of the baseline planner (baseline_p), DESPOT-α, and our proposal (recall_p).

Figure 6 summarizes our preliminary results for two known benchmarks in the literature of robot navigation with POMDPs, Tag and RockSample [2]. The recall_p (yellow lines) is sequential and brings little to no improvement in the planning time compared to baseline_p (blue lines). Both use the maximum time available to search for the navigation action (one second) and obtain indistinguishably "good" policies in most experiments.

Then we compare the results to other versions of the recall_p that implement the similarity search method with oneDPL. The recall_p_cpu implementation offloads the kernel to the multicore (green lines) and reacall_p_gpu to the integrated GPU (red lines). For the multicore implementation with oneDPL, we see a 2.5 to 5 × reduction in the planning time, enabling real-time performance for the evaluated benchmarks.

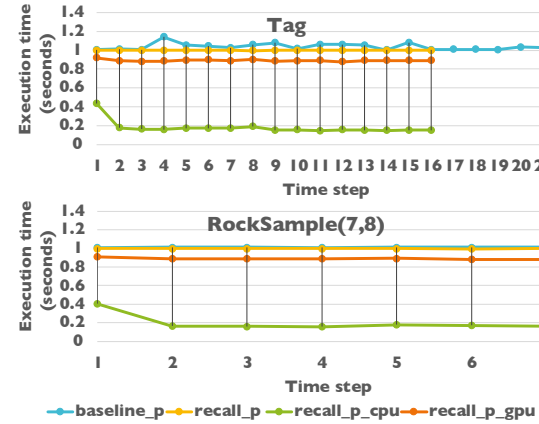


Figure 6. POMDP planning time evaluation for baseline_p, recall_p, recall_p_cpu, and recall_p_gpu on Tiger Lake platform. On the X-axis, we use as a timescale the planning timestep. Y-axis shows the planning time.

By carefully setting the experience memory parameters, recall_p may converge in fewer time steps and produce a superior policy. For example, we have the Tag benchmark evaluation in **Figure 6**, where the robot tags its target in 16 moves (time steps or actions) when using our experience memory data structure. In comparison, baseline_p requires 21 moves, given the same scenario and initial conditions.

SUMMARY OF RESULTS

During the first development phase, we have evaluated three different programming models, including oneAPI using the DPC++ programming language for planning sequences of actions for mobile robot navigation. When the scheduling strategy is carefully selected, we have found DPC++ to be five times easier to program while incurring only three to eight per cent of overhead.

In the second part, we take the lessons learned from optimizing Value Iteration for low-power execution and apply them to POMDPs—a more complex autonomous decision-making framework that accounts for all sources of uncertainty in the agent interaction with the environment. We propose a new method for online planning under uncertainty for POMDPs, Recall-Planner, that outperforms the state-of-the-art online planners for a set of benchmarks.

A novelty of our solution is that it allows us to plan for large problems on low-power SoCs, and we actively seek to achieve energy efficiency and not just make code run fast. Also, we have created a set of robot

navigation benchmarks for testing decision-making for basic navigation to a goal. The source code is available on bitbucket [13].

CONCLUSION

Memory and power are scarce for low-power embedded systems and mobile robotics computing. Despite this, applications often need close to real-time performance while balancing power use. In this work, we have presented some solutions for decision making in robotics using SoCs in low-power systems. We conclude that implementations based on oneAPI are more readable, shorter, and easier to debug. We think that a process or method that is compact, easy to understand and reproduce is far more valuable than a method that works slightly better at the cost of added complexity. In the short term, we hope this work will benefit developers who care about runtime performance and energy efficiency, as our ultimate goal is to make it feasible to develop intelligent agents and autonomous decision-making applications on mobile robots.

FUTURE DIRECTIONS

To date, we have covered the MDP category of decision-making problems and are working on POMDPs. Once we achieve this milestone, we will implement benchmarks on an AgileX Scout mobile robot. The resulting software will be made public on bitbucket.

ACKNOWLEDGMENT

This work was supported in part by grants from TIN2016-80920-R, PID2019-105396RB-I00, UMA18-FEDERJA-108, UMA18-FEDERJA-113, and Intel.

■ REFERENCES

1. Bellman R., "A Markovian Decision Process", Journal of Mathematics and Mechanics. vol. 6, no. 5, pp. 679–684, 1957. (journal)
2. Pineau J., Gordon G., Thrun S., "Point-based value iteration: An anytime algorithm for POMDPs", IJCAI, pp. 1025–1032, Aug. 2003. (conference proceedings)
3. Constantinescu D.-A., Navarro A. et al., "Performance evaluation of decision making under uncertainty for low power heterogeneous platforms." J. Parallel Distrib. Comput., vol. 137, pp. 119-133, 2020, doi:10.1016/j.jpdc.2019.11.009. (journal)
4. Nicholas R., Gordon G., Thrun S., "Finding approximate POMDP solutions through belief compression." J. Artif. Intell., vol. 23, pp. 1-40, 2005. (journal)
5. Schön T. B. "Estimation of nonlinear dynamic systems: Theory and applications", 2006. (Thesis or dissertation)
6. Reinders J., et al. "Data parallel C++: mastering DPC++ for programming of heterogeneous systems using C++ and SYCL", Springer Nature, 2021. (book)
7. Corbera F., Rodríguez A., Asenjo R., et al., "Reducing overheads of dynamic scheduling on heterogeneous chips", arXiv preprint arXiv:1501.03336, 2015. (PrePrint)
8. Voss M., Asenjo R., Reinders, J., "Pro TBB: C++ parallel programming with threading building blocks", New York: Apress., 2019. (book)
9. Processor Counter Monitor – PCM (Github). [Online]. Available: https://github.com/opcm/pcm (URL)
10. Dios A. J., Asenjo R., Navarro A., Corbera F., Zapata E. L, "High-level template for the task-based parallel wavefront pattern", 18th International Conference on High Performance Computing, IEEE, pp. 1-10, 2011. (conference proceedings)
11. Garg N.P., Hsu D., Lee W.S., "DESPOT-α: Online POMDP planning with large state and observation spaces", Robot.: Sci. Syst., 2019. (conference proceedings)
12. Constantinescu D.-A., Navarro A., Corbera F., et al., "Efficiency and productivity for decision making on low-power heterogeneous CPU+GPU SoCs." J. Supercomput., vol. 77, no. 1, pp. 44-65, 2021, doi:10.1007/s11227-020-03257-3. (journal)
13. Sourcecode (Bitbucket). [Online]. Available: https://bitbucket.org/corbera/vi-mdp/src/oneAPI/ (URL)

Authors Biographies

Denisa Constantinescu is a PhD candidate in Mechatronics at the University of Malaga and an oneAPI Innovator. She has been PI of two research projects and has collaborated on five other projects. Ms. Constaninescu has co-authored 2 book chapters, and 2 journal papers and 12 conference communications. She received the "2021 SCIE-ZONTA Award in Informatics" from The Spanish Computing Scientific Society (SCIE) and the "Intel Innovator Award" for the project "Efficiency and Productivity for Decision-making on Mobile SoCs" in 2020. Her research interests include heterogeneous programming models, optimization, decision making, and mobile robotics. Contact her at dencon@uma.es.

Rafael Asenjo is Professor of Computer Architecture at the University of Malaga. He obtained a PhD in Telecommunication Engineering in 1997. His research interests include programming models, parallel programming, heterogeneous computing, parallelization of irregular codes and energy consumption. He has participated in 19 research projects and two research contracts, published in 33 international journals indexed in the JCR, 11 contributions to IEEE and ACM "Core A" conferences, 4 Keynotes, 9 book chapters and 40 international conferences. He served as General Chair for ACM PPoPP'16 and as an Organization Committee member and a Program Committee member for several HPC related conferences (PPoPP, SC, PACT, IPDPS, HPCA, EuroPar, and SBAC-PAD). Along with Michael Voss and James Reinders, he co-authored the latest book on Threading Building Blocks (Pro TBB). He is a oneAPI Innovator, SYCL Advisory Panel member and ACM member. Contact him at asenjo@uma.es.

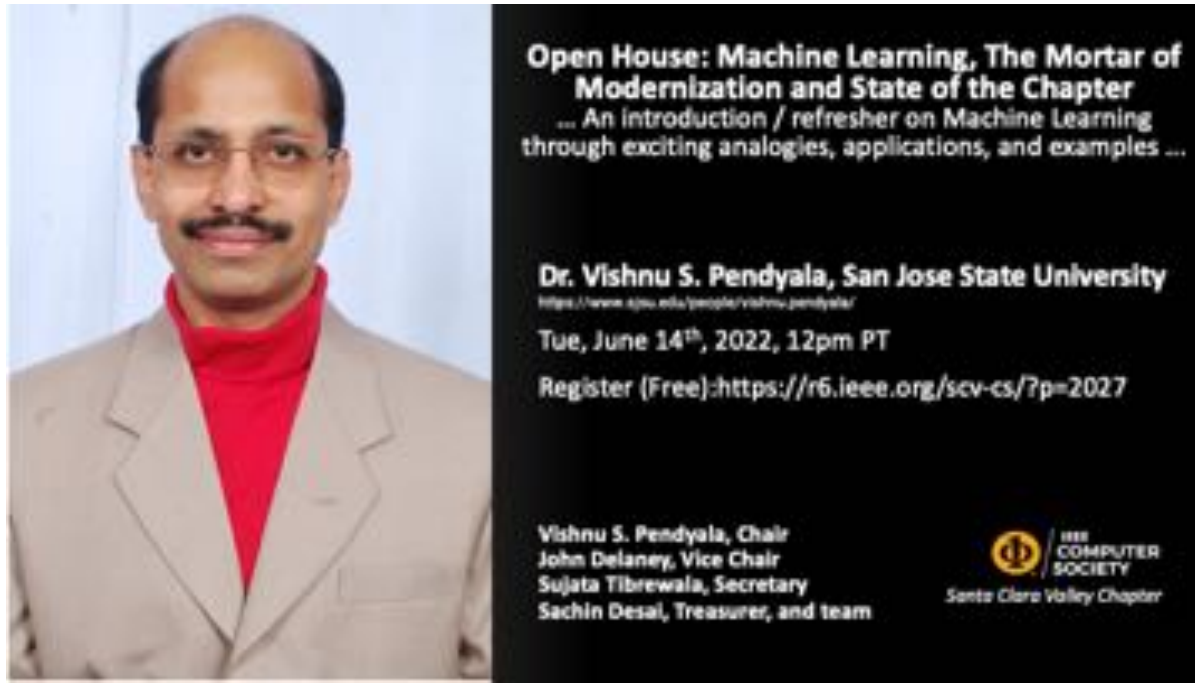
Angeles Navarro is full professor at the Department of Computer Architecture of the University of Malaga, Spain, since 2019. She received a PhD in Computer Science from the University of Malaga in 2000. She has been PI of several regional projects and has collaborated with national and European projects. Dr. Navarro has co-authored more than 90 papers, and 2 patents. She has served as a program committee member for several IEEE/ACM High-Performance Computing related conferences, as PPoPP, SC, ICS, PACT, IPDPS, ICPP, ISPA. She is the co-leader of the Parallel Programming Models and Compilers group at the University of Malaga.

Her research interests are in programming models for heterogeneous systems, analytical modelling, compiler, and runtime optimizations. Contact her at angeles@ac.uma.es.

Juan-Antonio Fernández-Madrigal is full professor at the System Engineering and Automation Dpt. of the University of Málaga. He received a PhD in Computer Science in 2000. He has worked on local, regional, national and UE robotics research projects since 1996, has 40 journal publications, 3 monographies, more than 70 conference communications and 6 patents. He has an H-index of 31 according to Google Scholar and has supervised a number of PhD theses on mobile robotics and cognitive robotics. His interests include mobile robotics, decision making, bayesian inference and cognitive robotics. Contact him at jafernandez@uma.es.

Ana Cruz-Martín is an associate professsor with the System Engineering and Automation Dpt. of the University of Málaga. She received the PhD in Computer Science in 2004. She has worked on local, regional, national and UE robotics research projects, which have led to several journal and conferences publications. Her research lines mainly involve educational robotics and machine learning techniques applied to mobile robotics. Contact her at acm@uma.es

Upcoming Event: Chapter Open House



Automatic Accident Detection Using Convolutional Neural Network and Internet of Things

Sathvika Kotha
CISCO, CX Group, Bengaluru, India.
sathvikakotha910@gmail.com.

***Abstract*— Accidents are a major cause of concern anywhere in the world. According to the road transport and highways ministry's report on Road Accidents in India, 2018, accidents are one of India's leading causes of mortality, accounting for around 64.4 percent of all deaths, and this number is steadily rising [1]. Identifying these accidents with the help of technology requires substantial research. This article mainly focuses on finding the best technology that is required for detecting traffic accidents. With the help of a 360 degrees surveillance camera, images such as automobile wrecks, blood, or a person laying on the road with no movement can be detected. This article discusses the technology and algorithm used to recognize and process images, as well as the technology that is utilized to alert hospitals for rapid assistance and to notify the victim's dependents or emergency contacts.**

1. INTRODUCTION

An accident is an unexpected event that typically happens suddenly and might result in injury, loss, and harm. Accidents usually involve human beings and lead to chronic disability or injury to a significant percentage of people in the world every year. Accidents may occur in various places, like at home, while traveling, in the workplace, in the hospital, or on the sports field. Many accidents lead to property damage or destruction. Many accidents can be avoided or prevented by taking proper safety precautions and being attentive to one's movements and surroundings. Ambulances and Hospitals play a significant role at the time of accidents. A victim's life is determined by how quickly we bring him or her to the hospital. We can save the victim's life if we get him or her to the hospital promptly; else, saving the victim's life can be extremely difficult.

2. PROBLEM DESCRIPTION

We know that Millions of people die every year in automobile accidents. According to the World Health Organization. (WHO), 1.35 million people die each year in car accidents around the world [2]. Victims of accidents may have a chance to live if they are transported to the hospital in an appropriate timeframe, but the greater part of them die because they are not sent to a medical center in a quick time. According to a BBC news article, after an accident in India, no one comes forward to help because people are afraid of being wrongly accused and of becoming stuck as a witness in a court case in India, where legal proceedings are famously lengthy [3]. Accidents might also happen in isolated or rural regions where no one will be around to contact an ambulance for assistance.

Scenario-1: Kanhaiya Lal begs out for aid, but automobiles swerve right past him. His young son and his wife and infant daughter's sprawled bodies lie next to the damaged motorcycle on which they had all been riding seconds before. Many Indians were disturbed by the extensively publicized CCTV film of this scenario, which showed the pain of a family of hit-and-run victims in northern India and the seeming indifference of passers-by. The family was finally rescued by some bikers and police, but it was too late for Lal's wife and daughter. Their killings triggered a national discussion about bystander involvement [3].

Scenario-2: The surveillance video clip shows a crowd of witnesses encircling Vinay, a 20-year-old boy, and doing nothing when he was hurled from his motorcycle in east Delhi by a fast automobile, and the clip [3].

Scenario-3: A man was driving to a construction site when he was hit by a car; as it was a distant location, no one was able to assist him because hardly anyone passes through that area.

3. PROPOSED SOLUTION

The street poles are equipped with 360-degree cameras. As mentioned in the block diagram in **Figure 1**, the surveillance camera and GSM/GPS module are connected to Raspberry Pi. When an accident occurs, a camera mounted on a street pole obtains a picture of the victim, which is then analyzed using image processing and CNN. It then delivers the images and notifications to the control room after it has completed its analysis. With the help of an application, the control room authorizes it and sends requests to the nearest hospitals. Once one hospital has accepted the request, the requests for the remaining hospitals are automatically terminated. The ambulance is dispatched to the site by the hospital that accepted the request. Following the victim's admission to the hospital, a call will be made to inform the victim's dependents about the victim's status and provide hospital information as can be seen in **Figure 2**. This method can be employed not only in the event of an accident but also in the event of an identifiable individual lying on the road due to health issues.

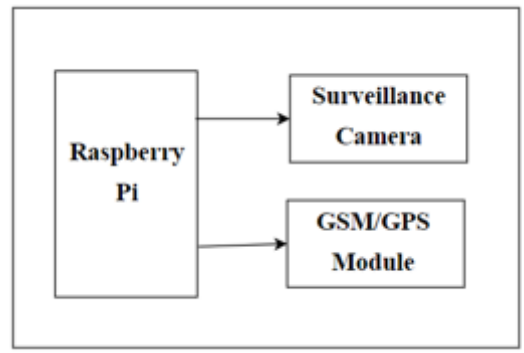


Figure 1. Block Diagram

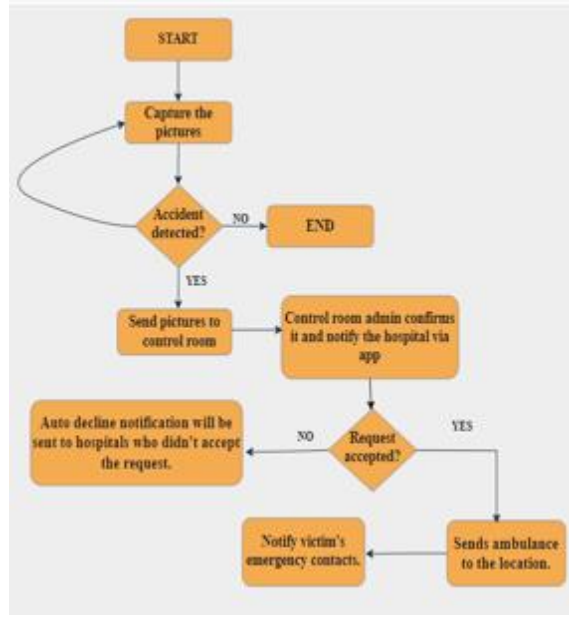


Figure 2. Flow chart of the proposed solution

4. INTERNET OF THINGS (IoT)

The Internet of Things (IoT) refers to the network of physical objects that are equipped with sensors, computing power, software, and other technologies to connect and exchange data with other devices and systems over the Internet or other communication networks.

4.1 Raspberry Pi

The Raspberry Pi Foundation, a UK nonprofit that aspires to educate people in computing and make computing education more accessible, has created a series of single-board computers known as the Raspberry Pi (see **Figure 3**). The Raspberry Pi was first published in 2012, and since then, various revisions and modifications have been developed. People across the world use Raspberry Pi to code, build hardware projects, home automation, edge computing,

and industrial applications. The cost of Raspberry Pi is low and runs on Linux operating system. It has a set of general-purpose input/output ports for connecting electronic modules or components and working on the Internet of Things (IoT). Initially Pi had a single-core processor 700MHz with 256MB of RAM, whereas the latest model has a quad-core 1.5GHz with 4GB of RAM.

Figure 3. Raspberry Pi

4.2 GPS Module

The tiny processors and antennas that the GPS modules are made of, receive data directly from satellites via specific radio frequencies. It will then get timestamps from all visible satellites, as well as other information. Arduino and Raspberry Pi support GPS modules (see **Figure 4**).

Figure 4. GPS Module

4.3 GSM Module

A GSM modem is also known as a GSM module. It is a physical device that uses GSM mobile phone technology to connect to a remote network. In the eyes of the mobile phone network, the GSM module is identical to a typical mobile phone. This module has a SIM to identify *itself* to the network. GSM modems provide TTL-level serial interfaces to their hosts. Embedded systems are the most common application for them. This module helps us in sending messages as well as provides the internet as it has a SIM (see **Figure 5**). If we connect this module to Raspberry Pi, we can write a python code and send messages or mail to the intended user.

Figure 5. GSM Module

5. DEEP LEARNING

Deep learning has had a significant impact on various domains of technology in recent years. One of the most talked-about technologies in the industry is computer vision. With the use of computer vision, computers can understand images and movies on their own. For example, computer vision is used in the automotive industry (autonomous vehicles), public security (facial recognition), transportation (violations detection), and traffic flow analysis. Image processing is the core of computer vision.

Image processing is the process of converting an image to a digital format and then executing specific operations on it to extract relevant information or an improved image. When applying predefined signal processing methods, the image processing system treats every image as a 2D signal.

Image processing broadly follows three steps:

1. Import the image with the help of image capturing software.
2. Evaluate and manipulate the image.

3. Report the result based on image analysis.

Deep learning is also a technique that mimics the human brain. In the 1950s and 1960s, researchers and scientists came up with a question: Can we make a machine learn and understand the data on its own like how a human learns and understands the data? This led to the invention of neural networks. The first simplest type of neural network is "perceptron". As it is not able to learn the data properly, In the 1980s, Jeoffrey Hinton invented the concept of Back Propagation. Many companies and people developed different efficient applications with the help of backpropagation and neural network architectures.

There are different types of neural network architectures:

1. Artificial Neural Network (ANN).
2. Convolutional Neural Network (CNN).
3. Recurrent Neural Network (RNN).

5.1 Artificial Neural network (ANN)

In **Figure 6**, the input is fed into the first layer which yields output to the neurons in the next layer and goes on which provides the result. The output is the result that is predicted either in 0 (non-accident) or 1(Accident). The neuron in each layer computes a function called the activation function. The activation function determines whether to activate the neurons to pass the signal to the neurons in the next layer.

The link between neurons of consecutive layers has a weight parameter associated with it. This weight impacts the output of each layer and the final layer output. Initially, the weights are assigned randomly and updated at each layer iteratively to yield the final output correctly. The main building blocks of neural networks such as a layer, neuron, weight parameters, activation function, and the optimizer will enable the neural network to select the suitable weight and produce the correct outcome.

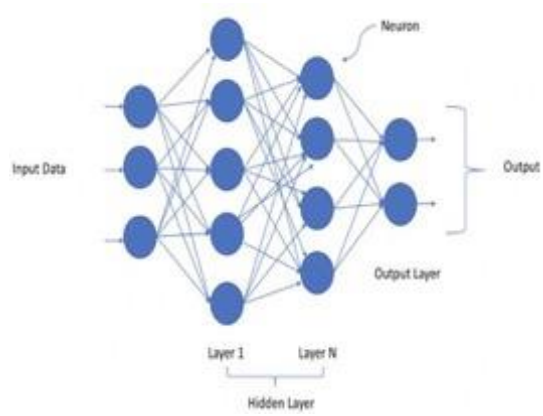


Figure 6. Artificial Neural Network

5.2 convolutional Neural Network

The convolutional neural network is a type of deep neural network. This kind of neural network is used when the input is in the form of an image. For instance, object detection, face classification, object classification, object recognition, and so on. Let's understand how we as humans understand and identify the image. The human brain is divided into four parts in which the cerebral cortex which is also known as the visual cortex is responsible for seeing the images and identifying them. CNN has multiple layers where each layer is responsible for performing each function. For instance, layer 1 is responsible for finding the edges of an image. Layer 2 is responsible to find if the object in the image is moving. Layer 3 is responsible to find if there are any other objects in the image and so on. In CNN these layers are known as filters. In image processing, black and white and gray-scale images are treated as 2D input, and color images are treated as 3D input, convolution operations are applied to those images to extract features using the backpropagation method.

In **Figure 7**, the input is given to the hidden layer. In each hidden layer, we perform convolution and activation functions on the image and send it to the next hidden layer, and so on. Once the convolution part is completed then the output is flattened which is the output of convolution layers gets converted into a feature vector for connecting it to the final layer.

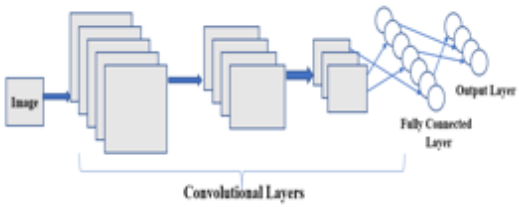


Figure 7. Convolutional Neural Network

5.3 Recurrent Neural Network

RNN is another type of neural network in which the output from the previous step is used as input in the next step. In general, all the inputs and outputs in standard neural networks are independent of one another. However, in some circumstances, such as when predicting the next word of a phrase, the prior word is necessary. Sentences are sequences of words, the previous words must be remembered. RNN was created for this purpose. It uses a hidden layer to remember certain information about a sequence, which is the most essential element of RNN.

6. ACTIVATION FUNCTION

The activation function is used in the process of converting the input signal to the output signal. It can decide whether to activate the specific neuron or not. The neural network could not learn complex network mappings without activation function as it is the one that converts the model to non-linear. The linear model has only limited capability and can't understand complicated images, speech, and so on. The activation functions are of two types: Linear activation function and Non-Linear activation function.

6.1 Linear Activation Function

This function resembles a linear line which doesn't work for complex parameters that are fed into neural networks (see Figure 8).

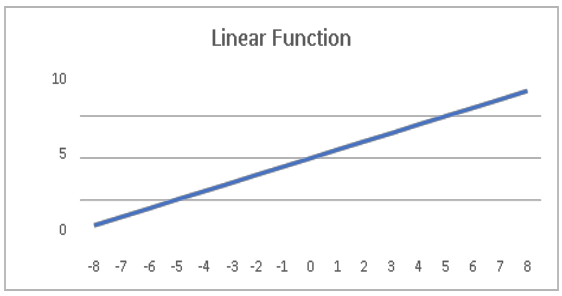


Figure 8. Linear Activation Function

6.2 Non-Linear Activation Function

Non-Linear activation functions are suitable for learning and generating complex network mappings.

6.2.1 Sigmoid Activation Function

The sigmoid function is used in the logistic regression model of machine learning, and it ranges between (0,1). It is also used in neural networks, and it ranges between (0,1) Sigmoid functions possess low convergence and vanishing gradient problems.

The sigmoid function is given by,

$$\emptyset(x) = \frac{1}{1 + e^{-x}}$$

Figure 9 shows the graph of the sigmoid function.

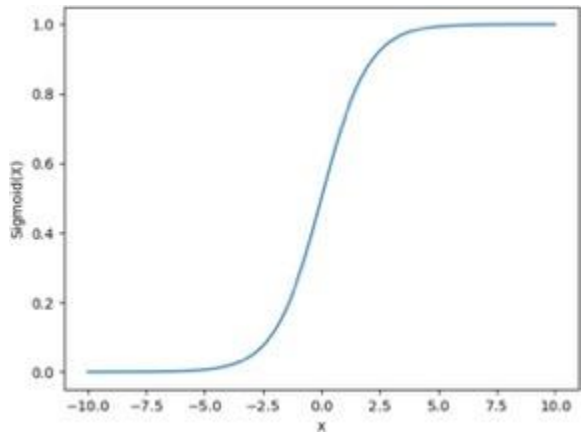


Figure 9. Sigmoid Activation Function

6.2.3 Rectified Linear Unit Activation Function

The most widely used activation function is Rectified Linear unit (ReLU). This function enables the mode to train with high performance. ReLU function provides better performance than the sigmoid function as it overcomes vanishing gradient problems (see **Figure 10**). This function does not activate all the neurons at the same time. The neurons get activated only if the output is greater than 0.

The ReLU function is given by,

$$f(x) = \max(0, x)$$

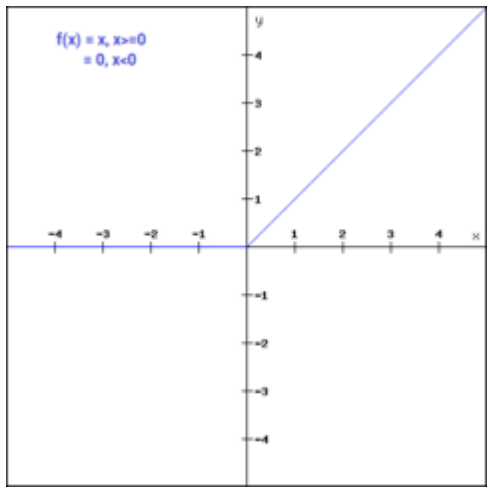


Figure 10. ReLU Activation Function

6.2.3 Softmax Activation Function

The Softmax function is used on the output layers of neural network models. This function predicts the probability distribution.

The softmax function is given by,

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

7. OPTIMIZER

The loss function is a function that helps in measuring if the model is working properly and providing accurate output. This function computes the difference between the value predicted by the model and the actual value. It is also known as the cost function or error function.

A common loss function is given by,

$$Loss = (y\hat{} - y)^2$$

In every epoch, we try to reduce the loss value. To reduce the loss value, we use optimizers.

7.1 Gradient Descent Optimizer

Gradient Descent will update all the weights in such a way to make the value of y^ similar to y. The learning rate is small so it takes much time to reach the global minima but if we take a higher learning rate value then the value will jump from one side to the other side and will never reach the global minima point. Here, we give all n data points once at a time as input to the neural network.

7.2 Stochastic Gradient Descent Optimizer

Stochastic gradient descent is similar to gradient descent but here we give only one record as an input to the neural network. If we give K number of records as an input, then it is known as mini-batch stochastic gradient descent. Gradient descent follows a straight path to reach the global minima, but SGD goes in a zig-zag way which takes more time to reach the global minima. There might be a chance of noisy data getting added while reaching the minimum point. SGD with momentum concept is used to alter the noisy data. An exponential moving average helps in removing noisy data.

7.3 Adaptive Gradient Descent Optimizer

ADAM is the mostly used optimizer in a neural network. The learning rate differs for each hidden layer and neuron. This optimization technique is more efficient and requires less memory.

8. IMPLEMENTATION METHODS

8.1 Using the CNN Model

A basic CNN model is used for implementation with specific batch size, image height, and width. Firstly, the data is divided into three: training, validation, and test dataset. Secondly, the images in the data set are preprocessed and then used these data to train the model. Finally, check the accuracy of the train data and validation data.

Table I shows us the amount of loss and accuracy of train data as well as validation data's loss and accuracy in each number of epochs. 5 epochs, ADAM as an optimizer, sparse categorical entropy as a loss function, and ReLU

as an activation function are used at the time of implementation.

Table 1. Loss and Accuracy of Train and Validation Data

Epoch	Loss	Accuracy	Val Loss	Val Accuracy
epoch 1/5	5.4577	0.5436	1.6529	0.4694
epoch 2/5	0.6433	0.6523	0.9371	0.5000
epoch 3/5	0.5999	0.6802	0.9210	0.4796
Epoch 4/5	0.5447	0.7143	0.7910	0.5510
Epoch 5/5	0.4574	0.7901	0.8555	0.5102

Figure 11 shows the accuracy of train data and validation data for a specific number of epochs and **Figure 12** shows us the loss of train data and validation data for a specific number of epochs.

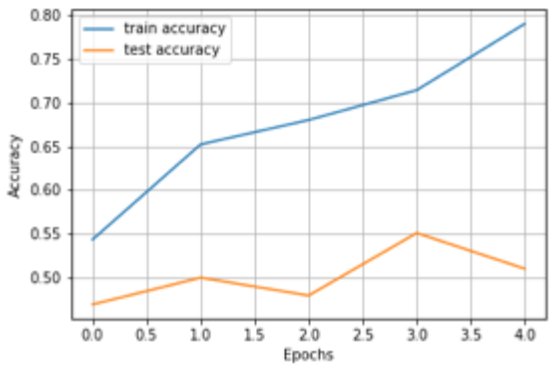


Figure 11. Train and Validation Accuracy

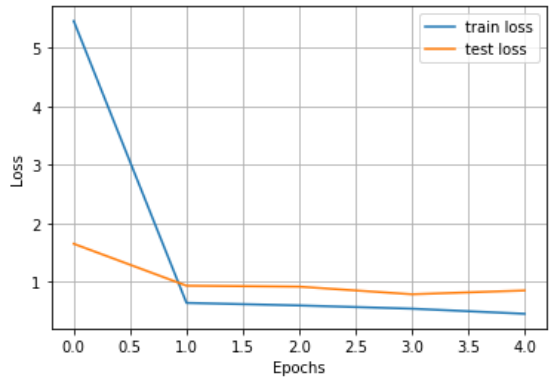


Figure 12. Train and Validation Loss

From Table 1, we can see that the accuracy rate of validation data is 51% but for better accuracy, we have totrain our model more properly so let's see the second implementation method which CNN is using the VGG16 model.

8.2 Implementation of CNN using VGG16 model and softmax function

In this method for the output layer, the softmax activation function is used and trained the model is with the training data set.

Table 2 shows the amount of loss and accuracy of train data as well as validation data's loss and accuracy in each number of epochs.

Table 2. Loss and Accuracy of Train and Validation Data

Epoch	Loss	Accuracy	Val Loss	Val Accuracy
Epoch 1/ 5	0.9948	0.5588	0.6791	0.6837
Epoch 2/ 5	0.5580	0.7143	0.3739	0.8061
Epoch 3/ 5	0.4254	0.8028	0.3343	0.8367
Epoch 4/ 5	0.4148	0.8129	0.2926	0.8776
Epoch 5/5	0.3346	0.8496	0.3051	0.8571

Table 3 describes the detailed VGG16 model summary using CNN. For hidden layers, ReLU and Max pooling was performed and SoftMax on the dense layer to get a better output.

Table 3. Model Summary

Layer(type)	Output Shape	Param#
input_1 (Input Layer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
Block2_conv1 (Conv2D)	(None, 112, 112, 64)	73856
Block2_conv2 (Conv2D)	(None, 112, 112, 64)	147584
Block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten_1 (Flatten)	(None, 25088)	0
dense_1 (Dense)	(None, 2)	50178
Total params: 14,764,866 Trainable params: 50,178 Non-trainable params: 14,714,688		

Figure 13 shows the accuracy of train data and validation data for a specific number of epochs.

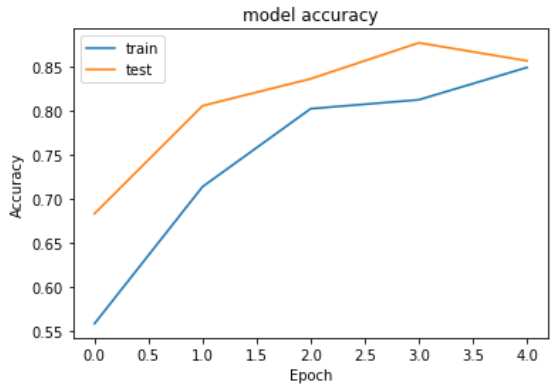


Figure 13. Accuracy of train and validation data

Figure 14 shows us the loss of train data and validation data for a specific number of epochs.

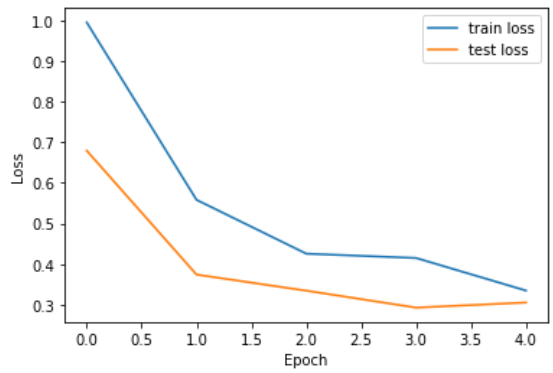


Figure 14. Train and Validation loss

9. LIBRARIES IMPORTED

9.1 Tensorflow

Tensorflow is an open-source platform for Machine Learning, Deep Learning, and Artificial Intelligence. It is designed in python programming language, so people feel easy to understand. It is a software library developed by google brain so the machine learning and deep learning concepts can be implemented easily [4].

9.2 Matplotlib

Matplotlib is an open-source library and was created by John D. Hunter. Matplotlib is a python visualization toolkit with a low-level graph plotting library [5]. Matplotlib is largely written in python for platform compatibility, with a few segments written in C, Objective-C, and JavaScript.

9.3 Keras

Keras was developed by Francois Chollet, and it is a deep learning framework for python. It supports different backends and platforms. It provides a python interface for artificial neural networks [6].

9.4 Pandas

Pandas is a Python package that allows you to work with large datasets. It offers tools for data analysis, cleansing, exploration, and manipulation. It was Wes McKinney who came up with the name "Pandas" in 2008. The word refers to both "Panel Data" and "Python Data Analysis."

9.5 NumPy

NumPy is a Python module for interacting with arrays. Travis Oliphant invented it in 2005. It also provides functions for working with matrices, Fourier transforms, and linear algebra. It is an open-source project, so is free to use. Numerical Python is referred to as NumPy.

10. DATASET USED

For the implementation, we used an accident-detection dataset from Kaggle [7]. As is commonly done, the Dataset is split into three parts: Training dataset, Validation dataset, and

Test dataset. Each dataset comprises two folders they are accident and non-accident (see Table 4).

Table 4. Dataset

	Accident	Non-Accident	Total
Training Dataset	369	422	791
Validation Dataset	46	52	98
Test Dataset	47	54	101
Total	462	528	990

11. ALGORITHM

1. Import required libraries like pandas, NumPy, Keras, Tensorflow, and so on.
2. Define batch specifications like batch size, image height, and width.
3. Load train data set.
4. Load validation data set.
5. Load test data set.
6. Define CNN using the VGG16 model.
7. Train model using train and validation data set. 8. Check accuracy.
9. Compare the loss and accuracy of train data and validation data using matplotlib graphs.

CONCLUSION

In this article, we could see that two implementation methods are followed. CNN using VGG16 gives better accuracy when compared to CNN. The loss in the second implementation method is gradually decreased. The accuracy of 85% is decent but if we would like to get more accuracy, then we can take big data set and work on it. Making model overfit is also not recommendable.

REFERENCES

[1] Nancy. P, Dilli Rao. D, Babuaravind. G, Bhanushree. S, "*HIGHWAY ACCIDENT DETECTION AND NOTIFICATION USING MACHINE LEARNING*", Vol. 9, Issue. 3, March 2020.

[2] "Road traffic Injuries" https://www.who.int/news-room/fact-heets/detail/road-traffic-injuries

[3] "If no one helps you after a car crash in India, this is why", BBC News, https://www.bbc.com/news/magazine-36446652

[4] "Tensorflow Documentation", https://www.tensorflow.org/

[5] "Matplotlib Documentation", https://matplotlib.org/

[6] "Keras Documentation", https://keras.io/

[7] "Accident Detection data set" https://www.kaggle.com/code/vanvalkenberg/cnn-for-accident-detection-83-val-accuracy/data

AUTHOR BIOGRAPHY

Sathvika Kotha received B.Tech degree in Computer Science and Engineering from Mallareddy Engineering College for Women. She is currently working at CISCO Systems Pvt Ltd. A network consulting engineer specializing in troubleshooting in Data Center storage and switching devices and have hands-on experience in automating networking applications using python, ML, web technologies, and virtualization. She is a certified Cisco Certified Network Associate (CCNA) and Cisco Certified Devnet Associate. Her research interests include automation using Machine Learning, Deep Learning, Data Science, and IoT.

A Lean Approach to Embedded Deep Learning

A. Gokhberg
FRAGATA COMPUTER SYSTEMS AG

***Abstract*—This article describes Arhat, a software framework providing innovative implementation for deployment of deep learning workflows. Unlike the conventional deep learning frameworks Arhat translates neural network descriptions directly into lean standalone executable code. Using Arhat, it is possible to generate executable code for each specific combination of a model and a target platform. This approach yields a very compact code and can greatly simplify the deployment process. Arhat supports a wide range of neural network operations and multiple target platforms including Intel CPUs and GPUs (via oneDNN) and NVIDIA GPUs (via CUDA/cuDNN or TensorRT inference library). Arhat is integrated with Intel deep learning software ecosystem including OpenVINO Toolkit and oneAPI Deep Neural Network Library (oneDNN). We have evaluated Arhat on Intel Tiger Lake i7-1185G7E and NVIDIA Jetson Xavier systems in the embedded object detection problem domain.**

■ **THE INTRODUCTION** Efficient deployment of modern deep learning solutions represents substantial challenges caused by high fragmentation of corresponding hardware and software ecosystems. This fragmentation makes deployment of each deep learning model a substantial engineering project and often requires using cumbersome software stacks.

Currently, the deep learning ecosystems represent heterogeneous collections of various software and hardware components, which include: (1) training software frameworks that produce trained models, (2) software frameworks designed specifically for inference, (3) exchange formats for neural networks, (4) computing hardware platforms from different vendors, and (5) platform-specific low level programming tools and libraries. These components are evolving at a very rapid pace. Compatibility and interoperability of individual components is frequently limited. There is a clear need for a streamlined approach for navigating in this complex world.

To address these challenges we have developed Arhat, a cross-platform framework for efficient deployment of deep learning inference workflows in the cloud and on the edge. Unlike the conventional deep learning frameworks Arhat translates neural network descriptions directly into lean standalone executable code.

In developing Arhat, we pursued two principal objectives: (1) providing a unified platform-agnostic approach towards deep learning deployment and (2) facilitating performance evaluation of various platforms on a common basis.

In this article we discuss design and architecture of Arhat and demonstrate its use for deployment of object detection models for embedded applications.

Reference architecture

Conceptual Arhat architecture is shown on **Figure 1**.

The central component is Arhat engine that receives model descriptions from various sources and steers backends generating the platform-specific code.

There are two principal ways for obtaining model descriptions. (1) Arhat provides the front-end API for programmatic description of models. (2) Pre-trained models can be imported from the external frameworks via bridge components supporting various exchange formats.

The interchangeable backends generate code for various target platforms. Backends for Intel, NVIDIA, and AMD hardware are currently available.

This architecture is extensible and provides regular means for adding support for new model layer types, exchange formats, and target platforms.

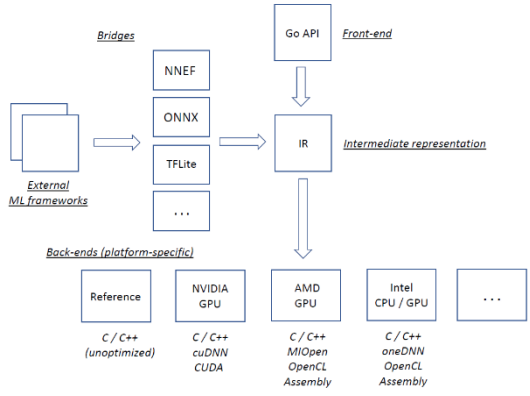


Figure 1. Arhat reference architecture.

Arhat core is implemented in pure Go programming language. Consequently, unlike most conventional platforms that use Python, Arhat uses Go also as a language for the frontend API. This API specifies models by combining tensor operators selected from an extensible and steadily growing operator catalog. The API defines two levels of model specification. The higher level is common for all platforms and workflows. The lower level is platform- and workflow-specific. The Arhat engine performs automated conversion from the higher to the lower level.

Support for Intel platforms via oneDNN

Intel oneAPI Deep Neural Network Library (oneDNN) [1] is a cross-platform library implementing basic deep learning operations optimized for various Intel CPU and GPU platforms.

Arhat supports Intel hardware via oneDNN backend. The architecture of this backend is shown on **Figure 2**.

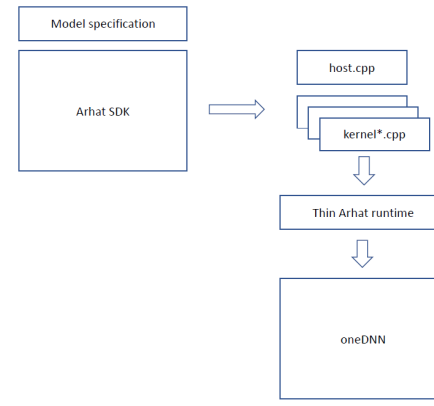


Figure 2. Arhat oneDNN backend.

The backend translates model specification into C++ code consisting of one host module and multiple kernel modules. This code runs on the top of a thin Arhat runtime that directly interacts with oneDNN. This approach results in a very slim deployable software stack that can run on any Intel hardware supporting oneDNN.

Interoperability with Intel OpenVINO

Intel OpenVINO toolkit [2] allows developers to deploy pre-trained models on various Intel hardware platforms. It includes the Model Optimizer tool that converts pre-trained models from various popular formats to the uniform Intermediate Representation.

We leverage OpenVINO Model Optimizer for supporting various model representation formats in Arhat. For this purpose, we have designed the OpenVINO IR bridge that can import models produced by the OpenVINO Model Optimizer. This immediately enables Arhat to handle all model formats supported by OpenVINO. The respective workflow is shown on **Figure 3**.

This development, on the one hand, integrates Arhat into Intel deep learning ecosystem and, on the other hand, makes OpenVINO capable of the native inference on platforms of different vendors. With Arhat, the end-users can view OpenVINO as a vendor-neutral solution. This might be beneficial for adoption of OpenVINO as an inference platform of choice.

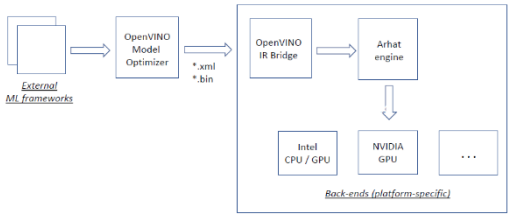


Figure 3. Arhat interoperability with OpenVINO.

Case study: object detection

We have validated the approach using as a case study object detection on embedded platforms. For this study we have selected 8 representative pre-trained models of 3 different families (SSD, Faster R-CNN, and YOLO) from Intel Open Model Zoo [3]. We used Arhat to generate code for Intel and NVIDIA and evaluated it on the respective embedded platforms.

We started with performance comparison of Arhat code powered by oneDNN against the open-source version of OpenVINO.

As a platform we used Tiger Lake i7-1185G7E, a system combining four CPU cores with integrated GPU. To obtain stable results and emulate embedded environment we have disabled turbo boost. We have measured average processing time for one image. Separate test runs have been made for the CPU cores and the GPU. As a baseline, we also included metrics for the ResNet50 image classification network.

Results for the CPU and the GPU are summarized in **Tables 1 and 2**. The numbers represent time, in milliseconds, required to process one input image.

Table 1. Performance evaluation on Tiger Lake, multi-core CPU

Model	Arhat	OpenVINO
resnet50-v1-7	23.5	23.0
ssd_mobilenet_v1_fpn_coco	397	401
ssd_mobilenet_v2	14.7	14.9
ssd_resnet50_v1_fpn_coco	563	568
faster_rcnn_inception_v2_coco	244	236
faster_rcnn_resnet50_coco	722	719
yolo_v2_tf	188	190
yolo_v3_tf	205	201
yolo_v4_tf	409	427

Table 2. Performance evaluation on Tiger Lake, integrated GPU

Model	Arhat	OpenVINO
resnet50-v1-7	10.1	11.3
ssd_mobilenet_v1_fpn_coco	144	125
ssd_mobilenet_v2	9.0	10.7
ssd_resnet50_v1_fpn_coco	195	183
faster_rcnn_inception_v2_coco	117	82.8
faster_rcnn_resnet50_coco	252	259
yolo_v2_tf	64.4	60.8
yolo_v3_tf	72.3	68.2
yolo_v4_tf	140	135

These tables demonstrate that Arhat with oneDNN backend provides inference performance compared to OpenVINO on both multi-code CPU and integrated GPU of Tiger Lake i7-1185G7E.

Interoperability with TensorRT

To run OpenVINO models on NVIDIA GPUs Arhat provides two alternative backends. These backends use CUDA Deep Neural Network library (cuDNN) [4] and TensorRT inference library [5] respectively.

The architecture of TensorRT backend is shown on **Figure 4**.

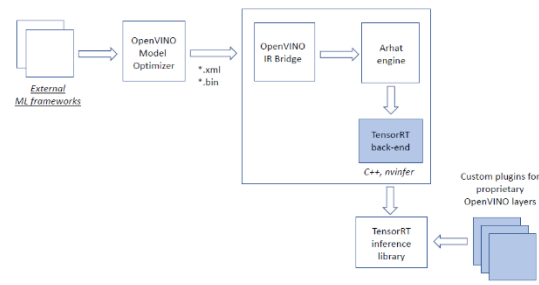


Figure 4. Arhat interoperability with TensorRT.

There are several OpenVINO proprietary layer types that are not directly supported by TensorRT. We have implemented them in CUDA as custom TensorRT plugins. These layer types include *DetectionOutput*, *PriorBoxClustered*, *Proposal*, *RegionYolo* and *RoiPooling*.

Evaluation on Jetson Xavier NX

As the next step, we included the NVIDIA platform in evaluation. We have chosen Jetson Xavier NX. This system has six ARM CPU cores and a GPU. It has various configurable power modes with 10W and 15W TDP envelopes and 2, 4, or 6 cores enabled.

The best performance is demonstrated with 15W and 2 cores, therefore we used this mode for comparison.

Results are summarized in **Table 3**.

Table 3. Performance evaluation on Jetson Xavier NX with Arhat cuDNN and TensorRT backends

Model	cuDNN	TensorRT
resnet50-v1-7	26.7	14.2
ssd_mobilenet_v1_fpn_coco	212	187
ssd_mobilenet_v2	23.4	12.3
ssd_resnet50_v1_fpn_coco	305	248
faster_rcnn_inception_v2_coco	195	159
faster_rcnn_resnet50_coco	413	366
yolo_v2_tf	114	83.3
yolo_v3_tf	139	94.8
yolo_v4_tf	250	191

This tables demonstrates that TensorRT inference library provides superior inference performance compared to cuDNN on Jetson Xavier NX.

Arhat backend for TensorRT enables native deployment of OpenVINO models on NVIDIA GPUs and opens a way for achieving the best OpenVINO performance on the hardware of the other vendors.

Conclusion

This study demonstrates that Arhat can efficiently interoperate with various key components of Intel and NVIDIA deep learning ecosystems. Furthermore, using Arhat allows to overcome limitations of each individual component and achieve results that are difficult or not possible to achieve otherwise. Arhat extends capabilities of Intel deep learning software by providing a way for the native deployment of OpenVINO models on the wider range of platforms.

Arhat can be also used for the streamlined on-demand benchmarking of models on various platforms. Using Arhat for performance evaluation eliminates overhead that might be caused by external deep learning frameworks because code generated by Arhat directly interacts with the optimized platform-specific deep learning libraries.

REFERENCES

1. oneAPI Deep Neural Network Library (oneDNN). [Online] Available: https://github.com/oneapi-src/oneDNN (URL.)
2. OpenVINO Toolkit. [Online] Available: https://github.com/openvinotoolkit/openvino (URL.)
3. Open Model Zoo repository. [Online] Available: https://github.com/openvinotoolkit/open_model_zoo (URL.)
4. NVIDIA cuDNN. [Online] Available: https://developer.nvidia.com/cudnn (URL.)
5. NVIDIA TensorRT. [Online] Available: https://developer.nvidia.com/tensorrt (URL.)

Alexey Gokhberg is a seasoned software engineer with more than 25 years of experience in various industrial and academic branches. His professional interests include deep learning, high-performance computing, programming language construction, and computational geophysics.

ICADS 2022

International Conference on Applied Data Science

IEEE COMPUTER SOCIETY

Santa Clara Valley Chapter

Call for Papers and Participation

IEEE Computer Society, Santa Clara Valley chapter is organizing a one-day International Conference on Applied Data Science (ICADS 2022). The conference will be held on Tuesday, July 12, 2022 and feature quick summaries of **previously unpublished** research, insightful experience reports, and demos in all areas of Applied Data Science. Short papers (3-5 pages) highlighting the work are invited from scholars all over the world. Submissions must confirm to the IEEE Conference formatting guidelines. Download template from https://template-selector.ieee.org/

Papers must be presented during the allotted time, virtually, in order to be included in the proceedings. Selected presentations will be broadcasted live on YouTube at mutually convenient times across the time zones. Proceedings will be published online and authors of select few papers will be invited to submit an extended version of their paper for inclusion in the future issues of the chapter's quarterly publication, Feedforward.

The conference will also include a few invited talks. Please contact vspendyala (at) hotmail.com with any questions or if you wish to help as a reviewer or in any other conference role. Authors, reviewers, and volunteers will receive a certificate of appreciation.

To submit your work, visit https://r6.ieee.org/scv-cs/?p=2034

Sign-up as a reviewer / volunteer: https://r6.ieee.org/scv-cs/?p=2039

Important Dates:

Submission deadline: Tuesday, May 31, 2022
Decision: Tuesday, June 14, 2022
Camera-ready copy due: Tuesday, June 28, 2022
Conference: Tuesday, July 12, 2022

Topics include

Data Mining
Machine Learning
Data Management
Big Data
Data Analytics
Deep Learning
Information Retrieval
Knowledge Graphs
Math for Data Science
Applications and more

Chair

Vishnu S Pendyala, PhD

Vice Chair

John Delaney

Secretary

Sujata Tibrewala

Treasurer

Sachin Desai

Webmaster

Paul Wesling

Connect with us

https://r6.ieee.org/scv-cs/
https://www.linkedin.com/groups/2606895/
http://listserv.ieee.org/cgi-bin/wa?SUBED1=cs-chap-scv&A=1
http://www.youtube.com/user/ieeeCSStaClaraValley