Adaptive Energy-aware Framework for Connected Vehicle Services
Approximate Computing for Vehicular Edge AI

Katare, D.

DOI
10.4233/uuid:05dfacb5-b2ab-4a17-a811-36185c091c70

Publication date
2025

Document Version
Final published version

Citation (APA)
Katare, D. (2025). *Adaptive Energy-aware Framework for Connected Vehicle Services: Approximate Computing for Vehicular Edge AI*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:05dfacb5-b2ab-4a17-a811-36185c091c70

Adaptive Energy-aware Framework for Connected Vehicle Services:

Approximate Computing for Vehicular Edge AI

Dewant Katare

Adaptive Energy-aware Framework for Connected Vehicle Services

Approximate Computing for Vehicular Edge AI

Adaptive Energy-aware Framework for Connected Vehicle Services

Approximate Computing for Vehicular Edge AI

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology
by the authority of the Rector Magnificus, prof. dr. ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates
to be defended publicly on
Monday, 8 December 2025 at 17:30

by

Dewant KATARE

Master of Science in Electrical and Computer Engineering,
Purdue University, USA
born in Tifra, Chhattisgarh, India

This dissertation has been approved by the promotors.

This project has received funding from the European Union's Horizon 2020 (H2020) Marie Sklodowska-Curie Innovative Training Networks H2020-MSCA-ITN-2020 call, under the Grant Agreement no 956090.

Keywords: Approximate Computing, Edge AI, Energy-aware Computing, Model Partitions, Resource Allocation, Edge Deployment

Cover by: Bahador Fatemi

Printed by: Print&Bind Amsterdam BV

ISBN 978-94-6518-199-8

An electronic copy of this dissertation is available at https://repository.tudelft.nl/.

Every element within nature's entirety approximates nature's full truth or the truth as we currently understand. Living with approximate answers to unanswered questions is far more fascinating than having incorrect answers. This thesis has approximate answers and a level of uncertainty about energy, accuracy and Edge AI deployments.
(Text inspired from Richard Feynman's, Lecture: Atoms in Motion, Caltech, 1963)

चतुराधिकं शतमष्टगुणं द्वाषष्टिस्तथा सहस्त्राणाम् ॥ अयुतद्वयस्य विष्कम्भस्य आसन्नौ वृत्तपरिणाहः ॥
Add 4 to 100, multiply by 8, and add to 62,000. This is approximately the circumference of a circle with a diameter of 20,000.
(Value of π approximated by Aryabhata in astronomical treatise 'Aryabhatiya')

Dedicated to my Parents Meera and Ram Kishore, with love and gratitude!

माता, पिता, गुरु, दैवम ॥

Contents

List of Figures

List of Tables

Summary

System developers and automotive manufacturers have proposed and used advanced driver assistance systems to enable the deployment of next-generation applications in connected vehicles, such as cooperative perception and vehicle-to-everything communication, while addressing autonomy-related challenges. The example deployment of these applications and systems, including proof-of-concept levels, generally depends on vehicle sensor suites, communication units, large-scale memory systems, and high-performance computing (HPC) units, which are together responsible for sensing the vehicle's environment, efficient data processing, and application deployment using machine learning models or rule-based algorithms. These models and algorithms are generally computationally complex as they process sensed and transformed data using statistical algorithms and deep learning models, such as those using convolutional operations and attention mechanisms, for tasks including decision-making, actuation, system analysis, environment sensing, monitoring, and infotainment applications. The computational complexity of these algorithms and models further increases upon scaling, primarily due to high data volumes, which also requires deep convolutions or similar operations for data processing. These operations require high-performance computing units to meet the application's operational and performance requirements, including latency, throughput, and accuracy.

Generally, the AI models used in connected vehicle applications are designed with a prime focus on model performance metrics; however, their high energy consumption and resulting carbon footprints are often overlooked. Recent studies have shown that the computing requirements for autonomous and connected vehicles can themselves become a significant component of overall energy consumption. For example, large-scale deployment of on-board AI across a global vehicle fleet could generate carbon emissions comparable to those of today's entire data center infrastructure. The computing hardware inside autonomous and connected vehicles can consume hundreds to over a thousand watts when running multiple perception and decision models simultaneously. Since these vehicles are battery-powered, this computing energy directly reduces driving range and increases operational cost. At fleet scale, this translates into a substantial carbon footprint, even when vehicles are electric. Therefore, energy efficiency is a primary design requirement, not only for sustainability but also for maintaining vehicle usability, battery longevity, and cost-efficiency. This thesis addresses the disparity between the strong research focus on model accuracy and the limited focus on energy usage by developing and evaluating an energy-aware adaptive framework for AI-driven vehicular services, such as non-safety-critical perception and high-definition mapping applications. The framework achieves energy and runtime improvements through energy-aware training, resource allocation, and adaptive deployment of computationally intensive models.

Previous research has proposed developing energy-efficient solutions in the hardware and software domains. For example, hardware-related energy-efficient solutions include transitioning from high-end graphical processing units to specialized AI accelerators and integrated circuits, which can process neural networks and related operations more efficiently. Similarly, architectural shifts and software-level solutions include transitioning from the centralized computing approach to dedicated edge computing and tiny machine learning solutions. However, the research scope remains within hardware-software co-design and optimization. By specifically targeting software-level optimizations, this thesis explores approximate computing (AxC) as a mechanism to utilize the error resilience of AI models in the perception and latency-tolerant applications of the vehicle-edge computing ecosystem. By balancing a trade-off between quality of experience and energy efficiency, AxC provides opportunities to reduce on-board energy demands and resulting carbon footprints of vehicle and edge devices while maintaining acceptable application performance levels. To explore and optimize the trade-off between model performance and energy consumption for connected autonomous vehicle applications, the following research questions are addressed:

1. What are the requirements for enabling energy efficiency in data-intensive vehicular services? **(RQ1)**
2. Which components can enable task deployments energy-efficiently and collaboratively in vehicle-edge-cloud computing? **(RQ2)**
3. How can energy-efficient components be integrated into an energy-aware adaptive software framework? **(RQ3)**
4. Can the framework effectively balance the trade-off between energy efficiency and performance in vehicle-edge-cloud computing scenarios? **(RQ4)**

Building upon existing research and knowledge on energy-efficient computing, this thesis addresses the above-mentioned questions. By addressing **RQ1**, this research identifies the technical and operational requirements to enable and integrate energy efficiency into data-intensive vehicular services. These functional requirements include performing high-level computations with minimal energy use and efficiently processing large data streams on edge devices. Secondly, to design and develop energy-saving components **RQ2**, the thesis proposes software-level approximation schemes combined with variational inference for both training-time and post-training model optimization and acceleration. Third, contributing to **RQ2** and **RQ3**, the research explores ML model partitioning and computing resource allocation mechanisms to utilize the distributed and heterogeneous nature of the vehicle-edge-cloud environment for distributed training and inference. These explorations aim to meet service-level objective deployment using lookup table-based mechanisms. Addressing **RQ3** and **RQ4**, the thesis integrates these components into an energy-aware adaptive software framework. This framework provides optimized model training and deployment strategies for distributed training and inference on heterogeneous computing resources, while effectively balancing the trade-off between energy efficiency and on-device application performance.

This thesis utilizes the design science methodology, adapting principles from the Information Systems Research Framework (design-as-a-search-process). This approach ensures research rigor to develop artefacts based on the application domain and existing theoretical knowledge. Further within the process, it adds design knowledge to the existing knowledge base of the application domain. As the development cycle of the research methodology includes tests and experiments, the effectiveness of the developed artefacts can be seen through test and experimental evaluation. Applying the proposed software approximation schemes, model partitioning, resource allocation, and adaptive deployment strategies on the state-of-the-art models shows up to 40% improvements in energy saving for less than 7% quality or model performance degradation when compared to the full precision and central computing methods. Software approximation schemes include the design of approximate multipliers, probabilistic approximation mechanisms, approximating convolutional, and fully connected layers for CNNs/DNNs. For the next-generation and memory/compute-intensive vision transformer models, this work proposes software-level approximation schemes based on variational inference, combined with post-training quantization and quantization-aware training, which show up to 35% improvements in energy efficiency for 6–8% quality loss. As the backbone of these next-generation vision models also includes multi-precision operands such as 8-bit, 16-bit, and 32-bit in layers and channels, the research also explores the advantage of mixed-precision operation to facilitate a balanced trade-off between models' energy usage and accuracy.

This research is among the first to investigate energy-aware requirements for application deployment beyond the traditional approach that generally focuses on cloud-based offloading mechanisms and model compression in the context of connected vehicle services and systems. The evaluation of the energy-aware framework on the popular edge devices shows the contribution of the thesis within the scope of distributed model computing using edge AI and sustainable computing practices. The research is set within the area of tiny machine learning and green AI principles. Future research can further develop adaptive algorithms that dynamically optimize energy use in real-time and investigate predictive models under varying conditions. Additionally, exploring the integration of Approximate Computing with emerging technologies like neuromorphic computing can improve processing efficiency in vehicular systems.

Samenvatting

Systeemontwikkelaars en autofabrikanten hebben geavanceerde rijondersteuningssystemen voorgesteld en toegepast om de inzet van volgende generatie toepassingen in verbonden voertuigen mogelijk te maken, zoals samenwerkende waarneming en voertuig-naar-alles-communicatie, terwijl zij omgaan met uitdagingen rondom voertuigautonomie. De voorbeeldimplementaties van deze toepassingen en systemen, zelfs op proef- of demonstratieniveau, zijn doorgaans afhankelijk van voertuigsensoren, communicatie-eenheden, grootschalige geheugensystemen en rekenhardware met hoge verwerkingscapaciteit (HPC-systemen). Samen zijn deze verantwoordelijk voor het waarnemen van de voertuigomgeving, het efficiënt verwerken van gegevens en het uitvoeren van toepassingen met machine learning modellen of regelgebaseerde algoritmen. Deze modellen en algoritmen zijn meestal rekenintensief, omdat zij waargenomen en getransformeerde gegevens verwerken met statistische methoden en diep-neurale netwerken, waaronder convolutionele bewerkingen en aandachtsschemas. Zij worden gebruikt voor onder andere besluitvorming, aansturing, systeemobservatie, omgevingswaarneming, monitoring en infotainment. De rekencomplexiteit van deze modellen neemt verder toe naarmate de schaal van gegevens groter wordt, wat leidt tot diepere convoluties of vergelijkbare bewerkingen voor gegevensverwerking. Dit vereist rekenhardware met hoge prestaties om te voldoen aan eisen zoals reactietijd, verwerkingssnelheid en nauwkeurigheid.

In veel gevallen zijn AI-modellen die in verbonden voertuigen worden gebruikt voornamelijk ontworpen met aandacht voor prestatieniveaus, terwijl hun hoge energieverbruik en de bijbehorende CO-uitstoot minder aandacht krijgen. Recente studies hebben aangetoond dat de rekenlast van autonome en verbonden voertuigen zelf een aanzienlijk deel van het totale energiegebruik kan vormen. Grootschalige inzet van on-board AI binnen een wereldwijde voertuigvloot kan bijvoorbeeld leiden tot CO-uitstoot die vergelijkbaar is met de uitstoot van alle huidige datacenters. De rekenhardware binnen autonome en verbonden voertuigen kan honderden tot meer dan duizend watt verbruiken wanneer meerdere waarnemings- en beslissingsmodellen tegelijk actief zijn. Aangezien deze voertuigen vaak op batterijen werken, vermindert dit energieverbruik direct de rijafstand en verhoogt het de operationele kosten. Op vlootniveau resulteert dit in een aanzienlijke ecologische impact, zelfs bij elektrische voertuigen. Energiezuinigheid is daarom een belangrijke ontwerpvoorwaarde, niet alleen voor duurzaamheid, maar ook voor bruikbaarheid, levensduur van de batterij en kostenefficiëntie. Dit proefschrift richt zich op deze ongelijkheid tussen de nadruk op modelnauwkeurigheid en de beperkte aandacht voor energiegebruik door een energiebewust adaptief raamwerk te ontwikkelen en te evalueren voor AI-gestuurde voertuigtoepassingen, zoals niet-veiligheidskritische waarnemingstaken en HD-mapping. Het raamwerk bereikt energiewinst en kortere verwerkingstijden door energiegericht trainen, toewijzing van rekenmiddelen en adaptieve uitvoering van rekenintensieve modellen.

Eerder onderzoek heeft energiezuinige oplossingen verkend binnen zowel hardware- als softwaredomeinen. Voor hardware omvat dit bijvoorbeeld de overgang van krachtige grafische rekenunits naar gespecialiseerde AI-versnellers en geïntegreerde circuits die neurale netwerken efficiënter verwerken. Aanpassingen in architectuur en software kunnen bestaan uit de verschuiving van centrale verwerking naar edge computing en kleinschalige ML-modellen. Toch blijft veel werk gericht op optimalisatie binnen ontwerp van hardware en software samen. Dit proefschrift richt zich specifiek op softwarematige optimalisaties en onderzoekt approximate computing (AxC) als een methode om gebruik te maken van de fouttolerantie van AI-modellen bij waarnemings- en latentie-tolerante toepassingen in voertuig-edge omgevingen. Door een evenwicht te vinden tussen gebruikerskwaliteit en energiegebruik biedt AxC mogelijkheden om het energieverbruik in voertuigen en randapparaten te verminderen terwijl prestaties op een bruikbaar niveau blijven. Om deze balans verder te begrijpen en te optimaliseren worden de volgende onderzoeksvragen behandeld:

1. Wat is nodig om energiezuinigheid mogelijk te maken in data-intensieve voertuigtoepassingen? **(RQ1)**

2. Welke componenten ondersteunen energiezuinige en gezamenlijke uitvoering van taken in voertuig-edge-cloud omgevingen? **(RQ2)**

3. Hoe kunnen deze componenten worden geïntegreerd in een energiebewust adaptief softwareraamwerk? **(RQ3)**

4. Kan dit raamwerk een balans bereiken tussen energiegebruik en prestaties in gedistribueerde voertuig- en edge-computing scenarios? **(RQ4)**

Door **RQ1** te behandelen worden de technische en operationele voorwaarden vastgesteld die energiezuinigheid mogelijk maken in data-intensieve voertuigtoepassingen. Deze voorwaarden omvatten onder andere het uitvoeren van complexe berekeningen met zo laag mogelijk energiegebruik en het efficiënt verwerken van grote datastromen op randapparaten. Voor **RQ2** stelt het proefschrift softwarematige benaderingsmethoden voor in combinatie met variatie-inferentie voor optimalisatie en versnelling van modellen tijdens en na training. Daarnaast onderzoekt dit werk modelpartitionering en toewijzing van rekenmiddelen in de gedistribueerde voertuig-edge-cloud omgeving om samenwerking tussen heterogene apparaten mogelijk te maken. Voor **RQ3** en **RQ4** worden deze onderdelen samengebracht in een energiebewust adaptief raamwerk dat strategieën levert voor gedistribueerde training en uitvoering, terwijl het de balans bewaakt tussen energiegebruik en prestatiekwaliteit.

Het proefschrift maakt gebruik van design science methodologie volgens het Information Systems Research Framework (design-as-a-search-process). Dit waarborgt methodische opbouw en toetsing van ontwikkelde artefacten in relatie tot bestaand domeinkennis. De effectiviteit van de ontwikkelde benaderingen wordt geëvalueerd via experimenten. Toepassing van de voorgestelde softwarematige benadering, modelpartitionering en adaptieve inzetstrategieën op state-of-the-art modellen laat tot ongeveer 40% energiebesparing zien met minder dan 7% prestatiedaling vergeleken met volledige precisie en centrale verwerking. Voor vision transformer modellen toont de combinatie van variatie-inferentie en kwantisatie energiebesparingen tot ongeveer 35% met 6% 8% prestatiedaling. Verder toont onderzoek naar mengvormen van numerieke precisies dat een gebalanceerde inzet van verschillende representaties kan bijdragen aan een betere afweging tussen energiegebruik en nauwkeurigheid.

Dit onderzoek is een van de eerste dat energiegerichte eisen voor toepassingsinzet onderzoekt buiten traditionele oplossingen die vooral gericht zijn op cloud-offloading of standaard modelcompressie in het domein van verbonden voertuigen. Evaluatie van het raamwerk op gangbare randapparaten laat de toepasbaarheid zien binnen gedistribueerde modeluitvoering en duurzame rekenpraktijken. Het werk sluit aan bij tiny machine learning en groene AI. Toekomstig onderzoek kan zich richten op adaptieve algoritmen die energiegebruik in real-time optimaliseren en op de mogelijke combinatie van approximate computing met neuromorfe hardware om efficiëntie verder te verhogen.

1 Introduction

Advanced driver assistance systems (ADAS) and in-vehicle services have evolved in recent years due to advancements in the vehicle sensor suites, computing units, automotive datasets, and software systems used for automating vehicular applications. This development has resulted in multiple levels of automation in current vehicles [1, 2]. The current trend is to integrate the sensors with state-of-the-art machine learning (ML) models, following the sense, think, and act mechanism widely used in robotics and automated systems [3, 4]. The goal of sensor and ML model integration is to assist or replace drivers by achieving the highest level of autonomy, which involves handling the entire driving process without human driver input [5–7].

Automotive manufacturers and system developers have already implemented driver assist features, which include automatic emergency braking, adaptive cruise control, GPS-based navigation, collision warning systems, obstacle detection, lane departure warnings, localization, high-definition maps and recently the inclusion of streaming and infotainment services to improve current driving applications and the quality of experience [2, 8–12]. Figure 1.1 shows the connected vehicle ecosystem, where vehicles and infrastructure sensors use inter- and intra-vehicle communication and distributed computing to process the sensed data using advanced machine learning models and computing mechanisms for collaborative application deployment, such as collaborative routing or platooning, traffic monitoring, connected infotainment applications [13–15]. These applications, supported by ML models, process sensed data to interpret the vehicle's surroundings, which helps to advance from the current vehicle assist services to provide next-generation on-demand mobility services, supporting the transformation of connected vehicles and intelligent transportation systems [1, 16–18].

The use of machine learning models in vehicles advances assist features and collaborative applications. However, their scaled deployment on vehicle computing units and edge devices introduces computing-related complexities, such as increased computing power requirements to process high-volume data, estimated to be approximately 20 Terabytes per hour for a vehicle [19]. This increased onboard processing requirement increases the energy consumption of computing and data processing [20]. With these scenarios, connected vehicles can be viewed as computers, or even datacenter on wheels, due to their advanced computing and communication capabilities [21, 22]. The demand for high-performance computing units in these applications results in additional energy consumption, creating a need for sustainable computing practices to mitigate the environmental impact [23–25].

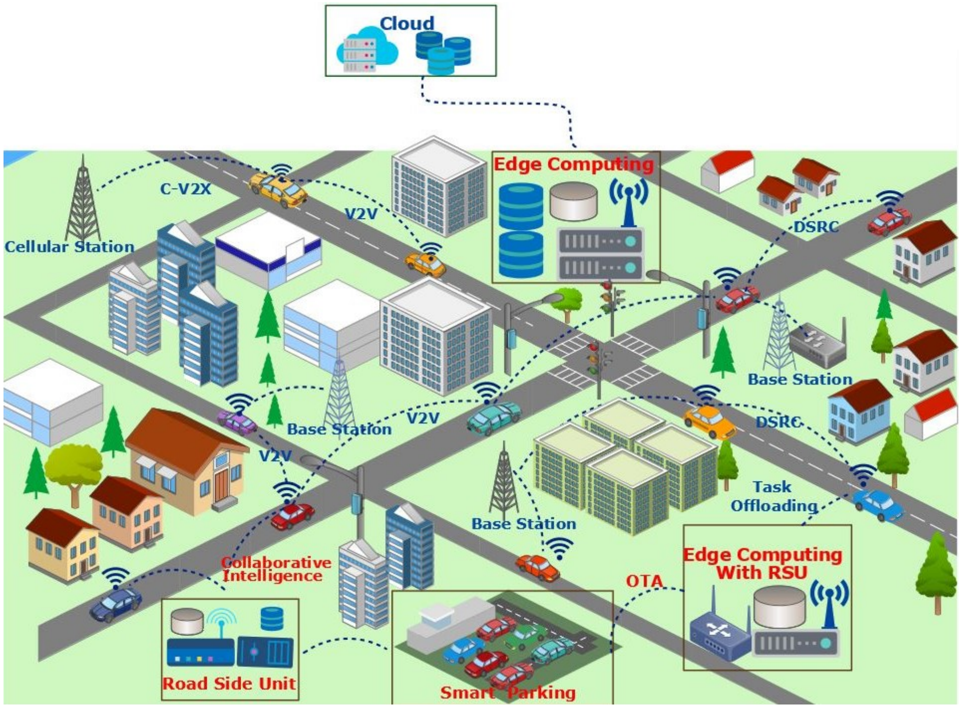


Figure 1.1: An overview of connected vehicle ecosystem (adapted from [20]).

Energy consumption and the carbon footprint of computing in connected vehicles were analyzed in the paper 'Data Centers on Wheels' by considering multi-modal scenarios [22]. The study considered vehicles with multiple levels of autonomy, currently used ML models, computing practices, and software systems and identified that data centers supporting connected vehicle services and applications could contribute approximately 0.3% of global greenhouse emissions, equivalent to the annual carbon emissions of Argentina [22]. The authors also modeled a scenario where one billion autonomous vehicles with a computing device consuming around 840 watts and operating for one hour will have greater energy consumption than current data centers, thus being an additional contributor to current carbon emissions. Based on the modeled scenario, the study described that the onboard computing power should operate under 1.2 kilowatts in over 90% of scenarios to keep emissions from autonomous vehicles lower than the data centres. This requirement shows the need for more energy-efficient and optimized hardware. The study also estimated a scenario where the assumption is that 95% of vehicles will be autonomous by 2050, and computational demands are expected to double every three years, which also requires global de-carbonization to progress, including further acceleration and optimization in hardware efficiency. This analysis shows the significant yet overlooked environmental impact of connected vehicles and associated computing practices from the perspectives of software development, hardware requirements, and efficiency. To systematically address computing-related challenges, the pyramid principle [26] is used for the articulation of computing-based energy efficiency issues in Connected and Autonomous Vehicles (CAV). In Figure 1.2, the first layer of the pyramid describes the connected vehicle domain, requirements and constraints, and unexpected results from the current computing practices. The second layer describes the scope and objective expected from this research. The third and fourth layer shows the computing approaches as a solution and their implementations within the edge AI subdomain. The last block covers application-specific approaches within distributed computing, model approximation and adaptive deployment. This structure and respective layers are derived from the literature review conducted to address the research question 1 (RQ 1). The remainder of this chapter covers approximate computing, edge AI, and the connected vehicle ecosystem.

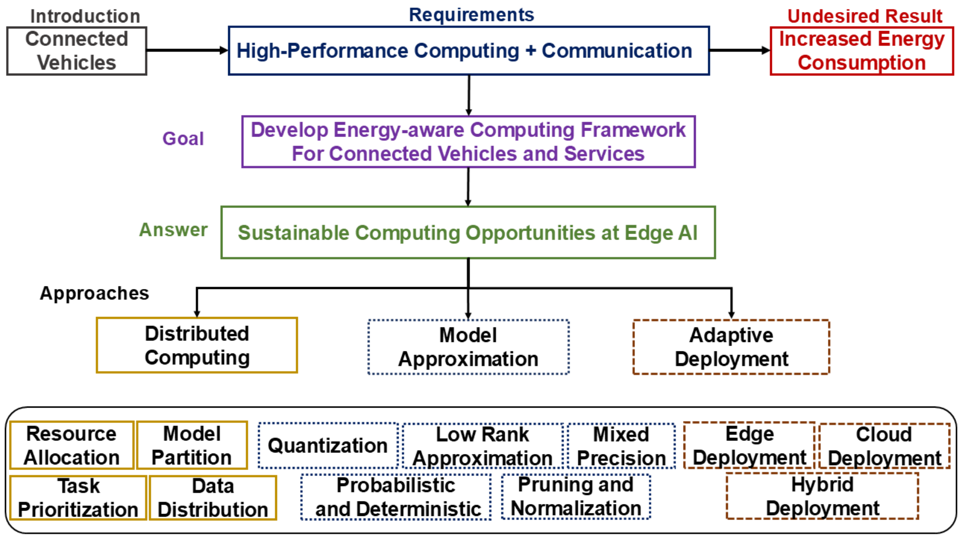


Figure 1.2: Computing and energy consumption issues in the connected vehicles explained through the pyramid principle [26].

1.1. Approximate Computing

The concept of approximation has been applied in the field of mathematics (partitions and derivatives), engineering (hardware-size reduction), and in computer science (vertex cover approximation algorithms and knapsack problems) as an alternative form of computation, equivalent representation or sometimes as an optimization strategy within functions, models and algorithms to address complexity challenges [28–30]. Because of these properties, approximate computing was proposed as an interdisciplinary area targeting hardware, software, algorithms, and models to perform a balanced tradeoff between metrics or to achieve multi-parameter optimization within a system [27, 31–33]. Examples of a few software-level approximations for models and algorithms related to this thesis are shown in the last block of Figure 1.2.

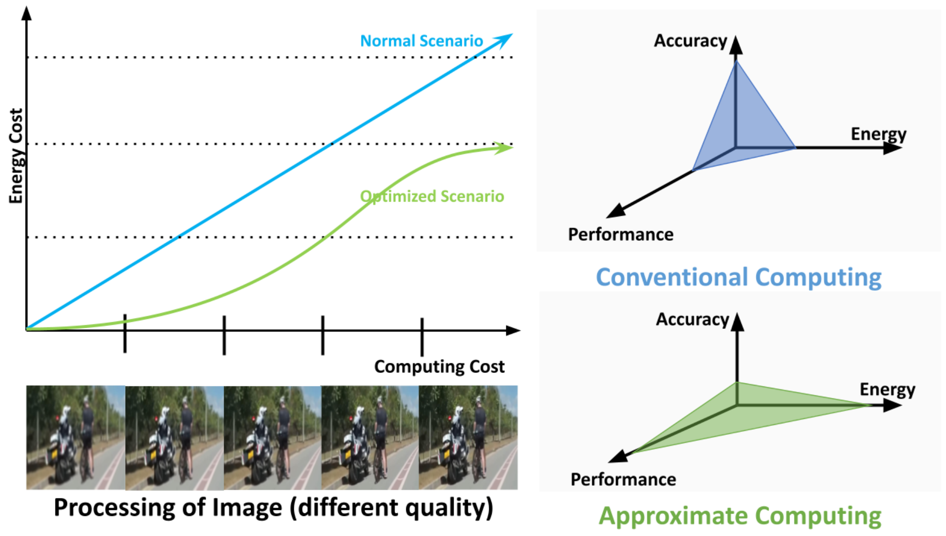


Figure 1.3: Computing cost and energy consumption in conventional and approximate computing while processing images of different quality [27].

Figure 1.3 shows a comparison between conventional (accurate and deterministic) computing and approximate computing for applications using the relation between accuracy, performance and energy efficiency. Conventional computing works on the principle of 'high accuracy', which requires sequential processing of *functions* and *data* to ensure that the application and task execution is predictable with a certain level of accuracy and system performance. However, approximate computing (AxC) is generally used for applications and systems that have error-tolerable properties, introducing controlled errors through techniques such as hardware voltage scaling, circuit simplification, software-based reduced precision, and dynamic pruning of non-essential computations [34, 35]. These approaches reduce computational load, and energy usage, and improve processing efficiency while balancing quality (accuracy) for applications such as multimedia, data analytics, and compression applications [27].

For machine learning applications, AxC has been used for designing AI accelerators, software and model-level approximation [36, 37]. Hardware approximation includes designing energy-efficient adders, multipliers and gates that have reduced form factor and perform computations with reduced precision or simplified architectures which deliver 'good enough' performance for specific tasks [27, 38]. Similarly, software approximation applies algorithmic adjustments or precision scaling to lower computational overhead and resource usage while maintaining acceptable accuracy [39, 40]. This may involve dynamically scaling the precision of calculations or using variational inference mechanisms that provide near-accurate results with resource-constrained computing units [41–43]. Approximation methods, categorized as linear, probabilistic, and deterministic strategies, are selected based on the application's error tolerance and operational requirements (Figure 3.1). Linear approximation converts complex (quadratic or exponential) computations into less demanding linear calculations [44, 45], while probabilistic approximation uses statistical approaches to provide results based on likelihood rather than precise calculations. Deterministic approximation simplifies computational problems without involving complex probabilistic methods, such as Laplace or mean-field approximations [43, 46].

Approximate computing addresses the challenge of high computational, memory requirements and energy demands associated with processing data from multiple vehicle sensors and communication systems for connected vehicle applications. By applying approximation techniques at the vehicle's onboard computing systems or at the edge processing units, connected vehicles can efficiently manage data processing requirements, reduce latency, and optimize energy usage. Adopting approximate computing aligns automotive technologies with sustainable goals for improved energy efficiency and optimized computing practices [47, 48], making it a strategic and balanced choice for future Edge AI supported vehicular applications.

1.2. Connected Autonomous Vehicles

Vehicle autonomy has been defined in six levels by SAE International (previously known as the 'Society of Automotive Engineers'), ranging from level 'zero' (no autonomy) to level 'five' (fully autonomous) [2, 20]. Mercedes-Benz became the first official automotive manufacturer in the year 2023 to receive state permission for deployment of SAE level 3 (conditional automated) driving vehicles in the USA [49]. Other manufacturers such as Tesla, Volvo, and Volkswagen are actively testing and refining ADAS technologies using software platforms also called (autopilot, drive pilot, FSD, etc.) to progress from the current level of autonomy to fully automated driving [2, 49, 50]. On the backbone, these software systems depend on the *sense-think-act* mechanism, commonly used in robotic systems [3, 4]. A description of the approach using automotive sensors and applications is shown in Figure 1.4. The approach involves capturing real-time data through vehicle sensors such as cameras, LiDAR, radar, GPS, and communication units including dedicated short range communication (DSRC). This is followed by data processing and feature extraction using AI models and regression algorithms, mainly deep neural networks. Such neural networks and algorithms perform essential driving functions, including object and obstacle detection, classification, localization, and path planning, influencing vehicles' actions in acceleration, steering, and braking tasks [7, 51]. These high-resolution sensors (LiDAR, camera, radar), together with onboard AI computing for real-time perception and decision-making, require energy-intensive hardware, such as high-performance GPUs and multi-core processors, capable of low-latency and high-throughput operation, which increases the computing load inside the vehicle. In hybrid and fully electric vehicle's, this load translates directly into additional power draw from the battery, ranging from tens of watts to several hundred watts or even over one kilowatt per vehicle for continuous perception and navigation tasks, which can reduce driving range and increase operational energy cost [22, 52].

The combination of advanced driver assistance systems (ADAS) with networking and communication technologies (vehicle-to-everything 'V2X') provides data transfer functionality from vehicles to infrastructure, other vehicles, and cloud platforms [1]. This

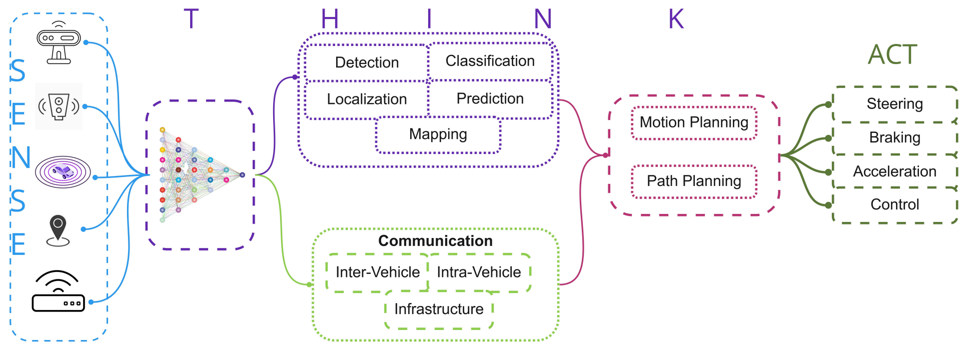


Figure 1.4: An overview of sense-think-act mechanism used in connected vehicles and other autonomous systems (adapted from [20]).

integration of connectivity and autonomy in the vehicular ecosystem allows the implementation of collaborative vehicular applications and services, which can be described as connected autonomous vehicles [7]. Examples of CAV applications are smart logistics, robotaxis, and mobility-as-a-service (MaaS) platforms to reduce human error and operational costs [53]. The motivation for application deployment is improved transportation efficiency through platooning (coordinated vehicle clusters), optimized traffic management via predictive algorithms, and improved vehicular safety through collision-avoidance systems [54–56]. From the scope of quality of experience, which includes high-volume data transfer and processing, popular applications proposed within CAVs using vehicle-edge-cloud infrastructure include high-definition maps, on-demand audio and video streaming, over-the-air updates, and edge analytics [18]. However, this deployment brings multiple computing challenges, mainly optimizing high-performance computing and large memory systems, increasing operational costs, and increasing the current energy consumption overall.

Deploying computationally complex AI models within automotive embedded systems amplifies the previously mentioned challenges. The increasing complexity of neural networks from CNN and DNN to currently proposed vision transformers and the requirement for real-time processing of sensor data exceeds the computational capabilities of conventional embedded platforms [20, 21]. Also, the existing data-management infrastructures in vehicles may not efficiently handle large-scale data volumes generated from high-resolution sensors, particularly under constrained energy and processing resources [57]. While edge computing and communication technologies can partially mitigate these challenges by offloading computational tasks to the cloud or adjacent edge devices, latency, reliability, and energy efficiency issues become a concern [58–60]. Additional complexities include thermal management for high-performance onboard computing units, cybersecurity vulnerabilities, and ethical considerations associated with AI-driven decisions. Thus, collaboration among automotive manufacturers, hardware designers, software developers, telecom providers, and energy management specialists is necessary to address these complexities and promote sustainable growth in intelligent transportation and connected autonomous vehicle ecosystems [23–25].

Table 1.1: Approximate number of sensors in a connected autonomous vehicle [20].

Sensors suites in vehicles based on autonomy levels					
Sensor	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Control Units	1	1	2	3	4+
Ultrasonic	5	5	9+	12	12+
Radar	2	4	6+	9+	12+
Camera	0	2+	8+	20+	20+
LiDAR	0	0	1+	3+	4+
GPS/GNSS	1	1	1	2	2
DSRC	0	1	1	1	1
V2X Module	0	1	1	2	3

1.3. Energy Efficiency and Edge AI

Fully connected vehicle ecosystem results from the collaboration of vehicles, vehicle and edge devices, edge servers, vehicle-edge-cloud communications and distributed computing systems (an example of the environment is shown in Figure 1.1). The current Level 1 to Level 3 vehicles have a few camera units, LiDAR, and multiple radars to assist drivers. The processing of sensed data is mostly carried out using a Graphics Processing Unit (GPU), and depending on the specifications, a GPU alone can consume up to 300–350Wh [24, 61] of energy per 100 km of driving, depending on the data rate and the quality of the sensors. As shown in Table 1.1, the number of sensors increases for fully connected autonomous vehicles compared to the current scenario; presented values are an approximate estimate depending on OEMs and fleets [61, 62]. Similarly, an estimate of energy consumption within a vehicle by its components (e.g., the embedded device running a DNN model, sensors such as LiDAR, and camera) is shown in Table 1.2 and the respective data rate from sensors is shown in Figure 1.5. This estimation is based on the current sensors suite and model deployment, which does not account for upcoming infotainment and user experience applications such as high-definition maps, mixed reality, on-board video streaming, and analytics.

Table 1.2: Energy estimates from vehicle components [20].

Source	Energy Consumption (%)
Computing units	(45 – 50)
Camera	(8 – 12)
Radar	(3 – 5)
LiDAR	(11 – 18)
Communication units	(5 – 10)
Auxiliary/Actuation/Power Electronics	(10 – 28)

Table 1.3 summarizes the data generation and communication demands across several vehicular services. These services vary widely in how frequently data is produced, how much information is transferred, and how quickly responses are expected on both

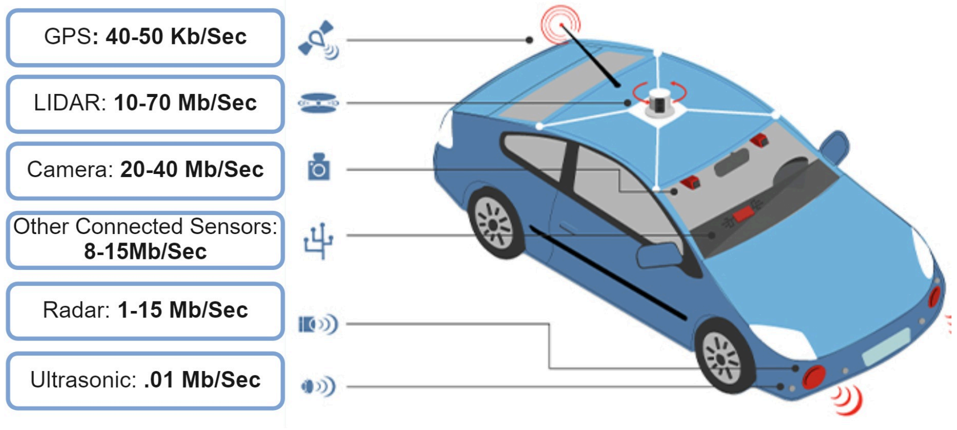


Figure 1.5: Approximate data-rate from the automotive sensors (adapted from [20]).

uplink and downlink channels. For example, V2Cloud Cruise Assist and Teleoperation depend on continuous video streams with short response time requirements, while HD map generation and digital twin updates involve larger batches of image and point cloud data transferred over longer intervals [18, 20]. Intelligent driving relies on ECU data that is generated in smaller bursts but still requires timely transmission for monitoring and coordination tasks. Together, these services highlight the range of computational, memory, and communication capacities needed to support connected and autonomous vehicle ecosystems.

Table 1.3: Data Generation and Requirements for Vehicular Services [18, 20].

System Requirements	**Data Source**	**Data in Vehicle EB/month**	**Target Data Traffic Rate**	**Uplink**	**Downlink**
V2Cloud Cruise Assist	Video stream	~ 1215	<10 sec	Continuous	Continuous
HD Map Generation	Image, Point cloud	~ 375	1–10 EB total	<1 week	<1 week
Intelligent Driving	ECU data	~ 22.5	<10 min	Occasional	Continuous
Teleoperation	Video stream	~ 583	<1 sec	Continuous	Continuous
Digital twin	Still image, ECU data	~ 369	<10 sec	Cont/Occ	Cont/Occ

The analysis in this section is based on several assumptions about how data is produced and processed in diverse connected vehicle settings. For video streams, one high-volume scenario considers a 10 Mpixel input with 3 bytes per pixel and a 1/4 lossless JPEG reduction, operating at 30 FPS over an average daily driving period of 30 minutes. Another large-scale scenario involves 2 Mpixel video streams using the same color depth and compression at 30 FPS, collected from six cameras running continuously for 8 hours per day across a fleet of 2.5 million connected cars. For still imagery, two patterns are taken into account: a continuous capture of up to four images per second during a one-hour daily trip, and a spatial capture where a 10 Mpixel frame with the same encoding is recorded every two meters over an average monthly travel distance of 1,000 km.

Communication throughput is modeled using automotive ethernet with an effective usable bandwidth of 100 Mbps × 1/3. This reflects protocol overheads, scheduling behavior, and real-world network utilization. To evaluate responsiveness, two timing measures are considered. The uplink response time refers to the duration between an image being captured on the vehicle and its appearance on the remote operators display. The downlink response time refers to the duration from when the remote operator issues a command to when the vehicle carries out the corresponding action. These timing measures are especially relevant for remote monitoring, teleoperation, shared autonomy, and cooperative maneuvering, where communication lag influences operational stability and safety envelopes.

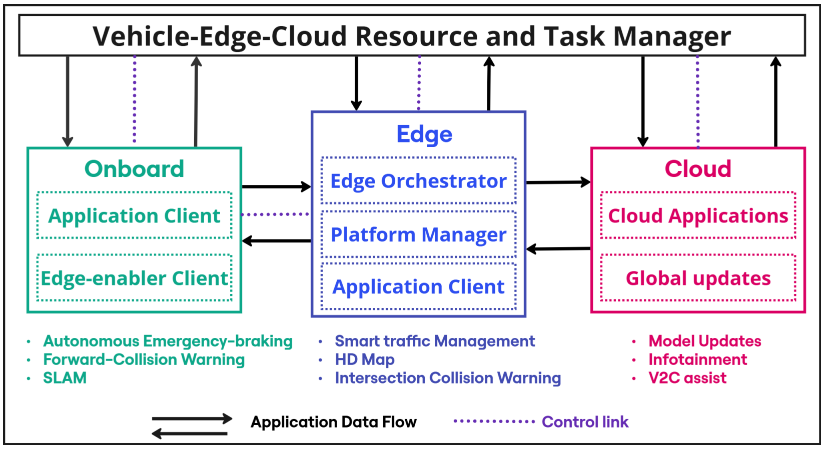


Figure 1.6: Application orchestration and data flow in vehicular ecosystem [18].

Together, these assumptions helps to compare data movement and processing needs across service types and for assessing how energy consumption scales with model size, runtime load, and communication frequency. The variability in data rates and timing expectations across these applications further emphasizes the importance of adaptive computing strategies that can adjust to workload, available resources, and deployment conditions. Figure 1.6 shows the vehicle-edge-cloud architecture inspired by the concepts proposed by the Automotive Edge Computing Consortium (AECC) [18], using the multi-layered approach for managing resources and tasks within vehicular networks. The architecture is described in three primary layers:

1. **Onboard Layer:** Incorporates both application and edge-enabler clients handling critical operational functions such requiring low-latency and real time processes, such as the autonomous emergency-braking system, forward-collision warning system, and SLAM (Simultaneous Localization and Mapping).

2. **Edge Layer:** Comprises the edge orchestrator and platform manager, facilitating immediate data processing tasks like smart traffic management and intersection

collision warning. This layer serves as an intermediary that processes data close to its source, optimizing response times and reducing data traffic to the cloud.

3. **Cloud Layer:** Hosts extensive data processing applications and model updates. Functions managed include global and local model updates, infotainment, and vehicle-to-cloud (V2C) assist services, which are essential for long-term system optimizations and updates.

Data and control flows between these layers ensure efficient task execution and resource management across the network. This setup provides a strategic allocation of computational tasks across the system, balancing immediate vehicle control needs with high-volume data processing requirements.

1.4. Stakeholders

The development of fully connected vehicles relies heavily on a broad ecosystem using communication and computation technologies, which are essential for handling and processing high volumes of data. The successful implementation of connected vehicular services depends on stakeholders' active participation and ability to effectively address the challenges of data transmission, computational demand, and energy consumption. Understanding stakeholders is necessary for identifying the various requirements, priorities, and constraints that influence the design and deployment of connected vehicular systems. Each stakeholder group, ranging from vehicle manufacturers and technology providers to regulatory authorities and end-users, contributes to varied expectations and objectives. Recognizing these perspectives helps guide the development of solutions that address technical challenges such as energy efficiency and data management, and also meet regulatory, safety, and user experience considerations. By mapping the influence and interests of different stakeholders, it becomes possible to anticipate potential conflicts, facilitate coordination, and design frameworks that are robust and adaptable in practical deployments. This understanding is also necessary for ensuring that emerging technologies are accepted, adopted, and maintained across the wider transportation ecosystem. Figure 1.7, shows a mapping of the vehicular ecosystem that is influenced by stakeholders with varying interests and levels of impact on energy-aware connected vehicular services.

The matrix categorizes stakeholders based on relative influence over system design and decision-making, and their level of interest in the outcomes of connected vehicular services. Stakeholder positions were determined through a review of existing automotive and edge computing ecosystems, including white papers and proof-of-concept documentation from industry forums [18, 20]. In this mapping, the car owner is placed near the center not because their influence and interest are equal, but because they generally have a high personal stake in cost, usability, and perceived benefit, while having limited direct influence over technical and regulatory decisions. Furthermore, regulatory bodies and automotive manufacturers hold greater influence, even if their interest levels may vary across specific service scenarios. Similarly, technology providers, energy regulatory groups, and mobility service integrators play key roles in shaping infrastructure capabilities and deployment practices. This positioning helps highlight where coordination

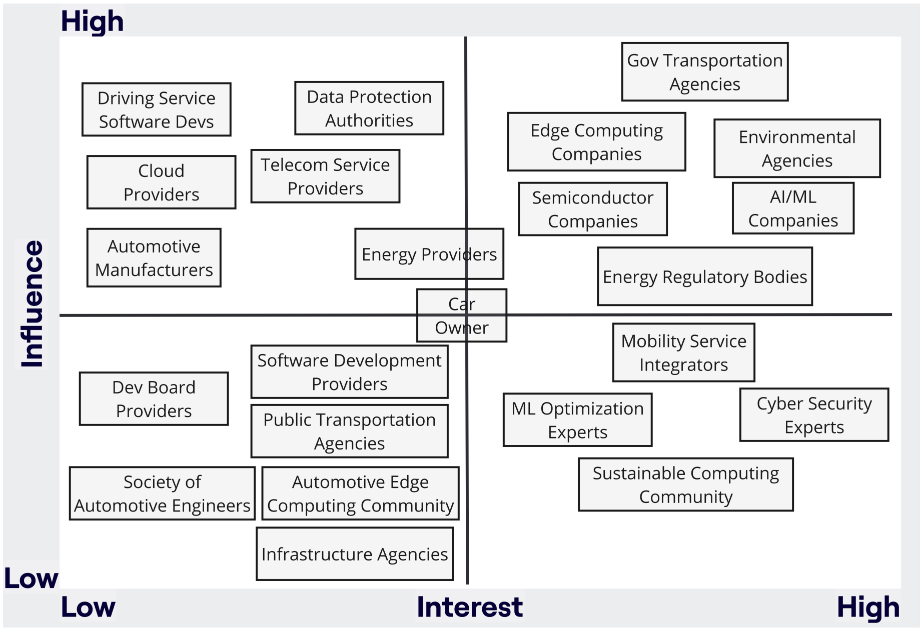

High
Influence
Low
Low
Interest
High
Driving Service Software Devs
Data Protection Authorities
Gov Transportation Agencies
Edge Computing Companies
Environmental Agencies
Cloud Providers
Telecom Service Providers
Semiconductor Companies
AI/ML Companies
Automotive Manufacturers
Energy Providers
Energy Regulatory Bodies
Car Owner
Mobility Service Integrators
Software Development Providers
Dev Board Providers
Public Transportation Agencies
ML Optimization Experts
Cyber Security Experts
Society of Automotive Engineers
Automotive Edge Computing Community
Sustainable Computing Community
Infrastructure Agencies

Figure 1.7: Categorizing stakeholders based on their influence and interest.

challenges are likely to arise and where alignment efforts are needed among technical, regulatory, and user-facing groups.

- **Driving Service Software Devs**: Develop software solutions that enable smarter driving services, significantly influencing automotive technologies.
- **Cloud Providers**: Offer cloud storage and computing resources that facilitate extensive data processing and management for vehicular systems.
- **Automotive Manufacturers**: Produce vehicles and integrate advanced technologies that define how automotive capabilities evolve.
- **Dev Board Providers**: Supply development boards that are crucial for prototyping and early-stage testing of automotive applications.
- **Society of Automotive Engineers**: Standardize practices and foster technology development in the automotive industry.
- **Data Protection Authorities**: Regulate and oversee data security and user privacy, ensuring compliance with legal standards.
- **Telecom Service Providers**: Provide the necessary network infrastructure to support connectivity for vehicle-to-everything (V2X) communications.
- **Energy Providers**: Supply power for general applications, including the operation of electric and hybrid vehicles and support infrastructure.

- **Software Development Providers**: Create software that enhances vehicle functionality and user interface.
- **Public Transportation Agencies**: Manage and integrate public transit solutions with automotive technologies for mobility services.
- **Automotive Edge Computing Community**: Develops and implements edge computing solutions that allow faster processing and responsiveness at the vehicle level.
- **Infrastructure Agencies**: Build and maintain physical and digital infrastructure crucial for modern transportation systems.
- **Edge Computing Companies**: Specialize in providing edge solutions that reduce latency and increase the efficiency of data processing near the source.
- **Semiconductor Companies**: Produce microchips and processors that power the advanced computing needs of modern vehicles.
- **AI/ML Companies**: Develop artificial intelligence and machine learning technologies that drive advancements in autonomous driving and vehicle intelligence.
- **Energy Regulatory Bodies**: Oversee and regulate how energy is used and managed within the automotive industry to ensure sustainable practices.
- **Mobility Service Integrators**: Combine various mobility services to provide coordinated transportation solutions, often utilizing advanced technologies.
- **ML Optimization Experts**: Focus on enhancing machine learning algorithms to improve their efficiency and effectiveness in vehicular applications.
- **Cyber Security Experts**: Ensure the security of vehicular systems against cyber threats, safeguarding data and operational integrity.
- **Sustainable Computing Community**: Advocates for and develops solutions that minimize the environmental impact of computing in automotive applications.
- **Environmental Agencies**: Monitor and regulate the environmental impact of transportation, pushing for greener alternatives and technologies.
- **Government Transportation Agencies**: Set policies and regulations that govern the operation and development of transportation systems, including vehicular technologies.

While considering the complex interdependencies among stakeholders in the vehicular ecosystem, it is necessary to recognize the trade-offs and resource re-distributions that affect overall system performance and costs. For example, enhancements in network capabilities by telecom providers might alleviate on-board computing demands, which can increase operational expenses and also require infrastructure upgrades across the network. Furthermore, differences and errors in data quality can lower the overall system and application performance, where a wrong input from one stakeholder could impair the functionality and effectiveness of connected services system-wide. Thus, promoting collaborative initiatives, such as joint development of energy-efficient technologies among automotive manufacturers, energy providers, and edge computing companies, is necessary (for example, working groups in the Automotive Edge Computing Consortium, the Edge AI Foundation, and the Edge AI and Vision Alliance). Such collaboration aligns with environmental regulations set by governmental bodies and also promotes the development and implementation of a sustainable vehicular ecosystem [63]. This interdependent approach ensures that improvements in one area do not negatively impact another, balancing energy consumption with computational demands and maintaining system integrity across diverse operational conditions. Overall, this diverse stakeholder environment describes the complexity of aligning various interests and roles towards a cohesive goal. While traditional roles have focused on specific aspects like network bandwidth or vehicle functionality, there is a growing requirement for a systematic approach to reduce energy consumption and optimize computational and communication demands. This research explores the potential of Edge-AI to address computing and energy challenges in vehicles and edge environments, aiming to enhance stakeholder awareness and foster collaborative efforts to develop energy-efficient vehicular services.

1.5. Knowledge gap and research questions

To identify the current research gaps and unresolved issues related to energy conservation in connected vehicles, a thorough literature review was conducted, focusing on the topics introduced in previous sections. This review primarily considered recently proposed AI and edge computing methods for connected vehicular services, with a particular focus on methodologies that address energy consumption, energy-saving techniques such as compression and reduction, collaborative and distributed learning, vehicular-edge computing frameworks, and model deployment on resource-constrained devices. The literature review [20] highlighted several knowledge gaps:

1. Lack of efficient data processing mechanisms tailored for energy saving and application optimization in data-intensive connected vehicular services. This gap suggests the need for new computational models that can handle large datasets effectively without excessive power usage.

2. There is a lack of designs that facilitate collaboration between vehicles and edge computing infrastructures to enhance energy efficiency in vehicular services. The stakeholders have different powers, control and interests, corresponding to data generation, processing location, communication and computing protocols. Enhancing collaboration between stakeholders could reduce energy and computing demands by optimizing data processing locations and tasks.

3. Current computing frameworks in vehicle-edge domain targeting energy can be categorized as energy-efficient, resource management, and data-compute offloading oriented. There is a requirement for an energy-aware adaptive framework that can dynamically balance the trade-off between energy consumption and computational accuracy in connected vehicular services. Such frameworks are essential for extending the deployment of intelligent vehicular technologies to resource-constrained environments.

Within connected vehicular applications, services can function even when there are small errors during processing. These tasks can be carried out with reduced precision, as long as the performance remains within an acceptable range for the application. Model compression and reduction-based approaches can be implemented for these tasks. However, this leads to the question “How good is good enough?”. Therefore, this research explores opportunities for on-board energy-saving by focusing on the methodologies from software-level approximate computing and Edge-AI. Based on the problem definition, the objective can be to design an energy-aware adaptive software framework that can be deployed on resource-constrained devices to implement data-intensive services.

Based on these gaps, the problem statement formulated is: **How can an adaptive, energy-aware framework be designed for integration on resource-constrained devices within connected vehicular services, and what requirements and challenges must be addressed to enable its practical deployment?**. Given the complexity of this question, which includes research design, modeling, and optimization, it is necessary to break it down into several sub-questions for a more structured research approach. Following the Design Science Research Methodology (DSRM) as outlined by Peffers et al. [64], the research process is organized around key activities: identifying requirements, designing and developing artefacts, and evaluating those artefacts in relevant contexts. In line with this methodology, the sub-questions are formulated to first understand and analyze existing knowledge base and computing practices in vehicular systems, then identify specific requirements and challenges for energy-efficient computing, followed by the design and development of energy-aware components, integration of these components into a practical framework, and finally, a thorough evaluation to show performance improvements and effective trade-offs. The sub-questions derived from the problem statement are as follows:

Question 1: What are the requirements for enabling energy efficiency in data-intensive vehicular services?

Answering this question will provide the technical and operational prerequisites needed for energy-efficient computing, such as AI models memory, and computational demands. The findings will serve as a foundation to explore suitable approximation techniques, model partition approach, resource allocation mechanisms, and adaptive model deployment at the edge that can be implemented within these parameters.

Question 2: Which components can enable task deployments energy-efficiently and collaboratively in vehicle-edge-cloud computing?

Exploring new software-level approximate computing schemes by identifying the computing elements and operations helps to characterize how different AI models supporting vehicular services in a heterogeneous setting can be approximated to optimize energy use. Understanding these interactions is crucial for designing an adaptive framework that utilizes these elements effectively.

Question 3: How can energy-efficient components be integrated into an energy-aware adaptive software framework?

This question focuses on integrating strategies derived from the previous answers to create a comprehensive system. It utilizes the independent energy-efficient solutions and *data gathered to process a reinforcement learning algorithm, converting these solutions to energy-aware approaches. This will lead to the development of a prototype energy-aware adaptive framework, which will then be tested for its impact on operational efficiency and energy consumption.*

Question 4: Can the framework effectively balance the trade-off between energy efficiency and performance in vehicle-edge-cloud computing scenarios?

This is an evaluation question; the answer to this question will assess the practical impacts of the framework, validating its effectiveness, efficiency, and tradeoff. The evaluation analysis further helps to refine the framework and areas of balanced tradeoff, while characterizing AI models and categorizing advanced optimization techniques.

1.6. Thesis Outline

Chapter 1 covers the introduction to energy-efficient solutions in connected autonomous vehicles (CAV) using Edge AI. It highlights the research's knowledge gaps, including a brief stakeholder analysis, the research goals and questions, and the research approach.

Chapter 2 covers the research method and core contributions covered in this thesis using artefacts, It also includes an overview of the guiding framework.

Chapter 3 covers background and fundamental concepts related to Approximate Computing and Edge AI, discussing their relevance to machine learning (ML) on resource-constrained battery-operated devices and embedded systems. The chapter elaborates on probabilistic AI models and the taxonomy of edge AI technologies for CAV.

Chapter 4 extends the discussion to issues of the model learning process using the currently available public datasets. The chapter examines the model learning process using metric-specific learning, which also helps to understand the weight distribution for approximation mechanisms within Edge AI, noting a knowledge gap in in-depth studies. It addresses the challenges of implementing current approaches and uses case studies to exemplify practical applications of solutions for bias detection in driving datasets.

Chapter 5 discusses proposed computational schemes for approximate computing, including approximate convolutions, approximate multipliers, and variational inference with mixed-precision quantization for energy-efficient training. The chapter presents case studies on adaptive deployment for edge computing, and perception in connected vehicles, exploring how these schemes affect model accuracy, energy, and overall system efficiency.

Chapter 6 discusses proposed model partition mechanisms and adaptive deployment strategies in distributed vehicle edge ecosystems. The chapter also includes a case study using serverless edge computing techniques.

Chapter 7 covers an overview of the proposed energy-aware adaptive framework for integrating approximate computing within edge AI systems. Discusses design considerations, including the necessary adaptations for dual or multi-modal systems and implementation challenges. The chapter evaluates the effectiveness of software approximation techniques and their impact on energy efficiency and system accuracy.

Chapter 8 summarizes the findings and contributions of the research. Reflects on the limitations of the current study and outlines potential areas for further research, providing recommendations for future work to advance and refine the integration of approximate computing and edge AI in autonomous systems.

2 Research Method

This research adopts the information systems research principle [65], to systematically design artefacts tailored to the application domain and enhance the existing knowledge base by adding and analyzing design knowledge. The principle is structured around three components: Environment, Information Systems (IS) Research, and Knowledge Base, as depicted in Figure 2.1. At the core, the **IS Research** consists of the design and evaluation of artefacts based on the needs and requirements of the application domain (from the environment), existing foundations and methodologies (from the knowledge base). In this context, the energy-aware adaptive software framework development within this research includes the design of components, prototype development, evaluation, and a set of guidelines for implementation. These design and development processes are fundamentals of Information Systems research [64]. The **Environment** consists of the problem space, and originally it contains people, organizations and technologies. For scoping this research, the technical requirements are outlined within the **Application Domain**. The **Knowledge Base** consists of existing research and scientific work in the form of theories, foundations, and methodologies, which also provide implementation processes and guidelines.

2.1. Research Approach

As shown in Figure 2.1, the process also includes three iterative cycles: Relevance cycle, Design cycle and Rigor cycle. **Relevance cycle** acts as bridge between the application domain and design process, focusing on the application's requirements and needs. **Rigor cycle** connects the design process to the knowledge base, integrating relevant knowledge and theoretical frameworks. **Design cycle** is an iterative process enabling the development of artefacts and evaluations within the research. Following this method, the research questions are strategically aligned with Hevner's principles in Figure 2.1. This structured approach ensures the research questions are integrated into Hevner's principle, which details the interconnections between the research questions and the design components and cycles.

2.2. Research Artefacts and Contributions

Design artefacts, as outlined in the design science research methodology [65], emerge from the **design cycle**. According to Artifact Typology [66], the anticipated artefacts from

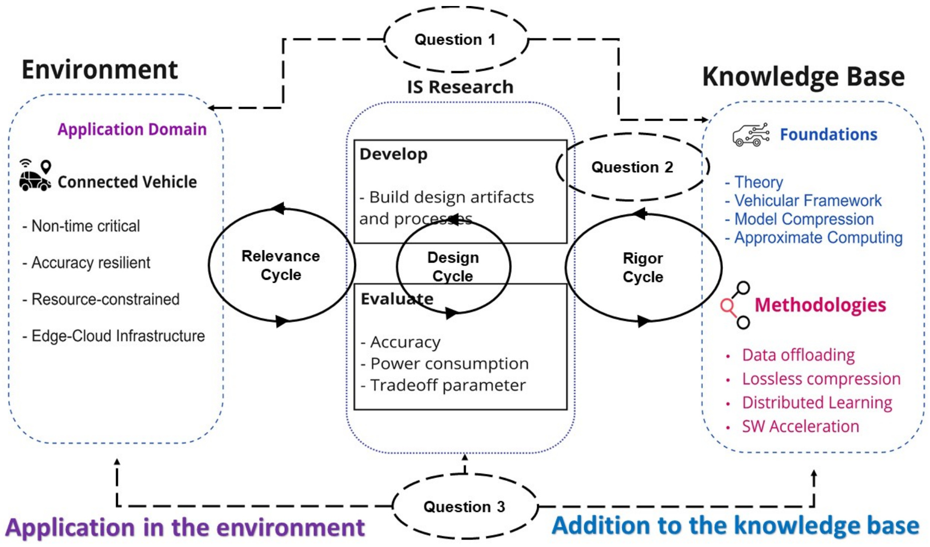


Figure 2.1: Research method adopted from Hevner [65].

this PhD project are categorized as follows: 1) **Requirement**, which describes necessary capabilities or functions for a system or application domain. 2) **Algorithm**, which provides executable processes for system behaviour. 3) **System Design**, which offers descriptions of a system's structure or behaviour, typically through formalism.

Artefacts have been broadly classified [66, 67] into several types of research outputs: **Constructs**, which are concepts used to describe problems within an application domain; **Models**, which are sets of propositions that establish relationships among concepts; **Methods**, which include algorithms or guidelines for accomplishing tasks; and **Instantiations**, which are realizations of an artefact within a specific environment. Particularly in engineering research, activities related to these artefacts involve building an artefact to perform specific tasks and evaluating the artefact based on predefined metrics and parameters. These activities typically occur in a loop and through iterations, aiming to enhance performance. In this research, the design artefacts are essential components leading to the design of an energy-aware adaptive software framework that incorporates advanced software, algorithmic, and communication components. This framework (section 2.4) is structured and detailed, with implementation guidelines tailored for the targeted application domain. An overview of artefacts developed in this research is shown in table 2.1. Each artefact developed in this research is considered from the perspective of achieving independent energy-efficient solution, and the artefacts also have interaction and sub-dependency to create a detailed energy-aware adaptive framework. The relationships between artefacts can be summarized as follows:

Requirement and System Design: The dynamic model partitioning requirements directly influence the system design. The design integrates these requirements to ensure the system can dynamically allocate resources and handle computational tasks efficiently.

Algorithm and Method: Algorithms for dynamic partitioning and adaptive deploy-

Table 2.1: Research Artifacts that are developed in this PhD Research.

Artifact Type	**Definition**	**Outcome**
Requirement	Describes necessary capabilities or functions for a system or application domain.	Dynamic model partitioning requirements for energy efficiency in vehicular services.
Algorithm	Provides executable processes for system behavior.	Algorithms for dynamic model partitioning and adaptive deployment to optimize computational tasks.
System Design	Offers descriptions of a system's structure or behaviour, typically through formalism.	System design for an energy-aware adaptive software framework integrating model partitioning and approximation schemes.
Construct	Concepts used to describe problems within an application domain.	Concepts such as "energy efficiency," "model partitioning," and "adaptive deployment."
Method	Includes algorithms or guidelines for accomplishing tasks.	Methods for applying linear approximation and probabilistic techniques to reduce computational demands.
Models	Computational representations that simulate or predict system behaviours based on varying parameters.	Approximated models developed through various approximation techniques; these models simulate the trade-off between energy consumption and computational accuracy under different operational conditions.
Instantiation	Realizations of an artifact within a specific environment.	Prototype development that integrates all strategies and methods into a functional system for testing and evaluation.

ment operationalize the methods developed for resource management and approximation. These algorithms implement the theoretical methods in practical testbed settings, ensuring optimal resource usage and energy efficiency.

Constructs and Models: Constructs such as energy efficiency and model partitioning frame the development of models that simulate various operational scenarios. These models use the constructs to predict the outcomes of different energy and accuracy trade-offs.

Instantiation and System Design: The instantiation of the framework as a prototype shows the overall practical application of the system design. It allows for testing and evaluation within the targeted environment, to show the framework's effectiveness in varied scenarios.

Artefacts Contribution: The key contributions from the artefacts and individual components point of view can be summarized as follows:

A) Adaptive Model Partition and Deployment: The scope of this component is to optimize computational tasks dynamically across the vehicle's onboard processors and edge devices to enhance energy efficiency with a traded computational performance. This involves

1. Dynamic Model Partitioning: Developing resource-aware cost-based analysis that assesses model computing properties to decide the best computation point onboard or at the heterogeneous edge.

2. Adaptive Deployment: Adjusting the computational load based on the computing device's operational status to utilize resources effectively and reduce energy consumption. The optimized implementation of these strategies will allow resource-constrained devices to handle complex computing tasks more efficiently, using optimal energy while ensuring that essential model functions are performed.

B) Model Approximation Schemes: Here, the focus is on developing and applying approximation schemes that reduce the computational demands of complex AI models. These schemes include:

1. Linear Approximation: Simplifying bit-multiplication operations within the models to decrease computational complexity and power usage.

2. Probabilistic Techniques: Developing statistical methods that provide or meet application-level good enough accuracy with a higher energy saving, suitable for non-critical tasks, and higher latency operations.

These approximation schemes are designed to balance computational accuracy and energy use, making them ideal for extended operation in the resource-constrained and energy-limited settings.

C) Bias Evaluation and Mitigation: Model approximation and compression strategies involve varied model training strategies, and these approaches may impact the trained weights, especially for samples and classes with varied presence during model training. Here, a critical research gap remains in understanding and mitigating these effects. Therefore, we perform bias evaluation and mitigation to enable rigorous evaluation and exploration of bias amplification in compressed models.

1. Bias Analysis: Using behavioural metrics for AI models to understand model learning process in the presence of biased data or class samples with disparities.

2. Bias Mitigation: Data sampling and mitigation strategies are used to test biases, and have a comparative study using behavioural metrics and cost-sensitive learning.

D) Energy-aware Design: This design integrates the independent energy-efficient techniques i.e. model approximation methods, adaptive model partitioning and deployment to collaboratively balance energy consumption and computational accuracy. The integration includes:

1. Algorithm Integrations: These algorithms incorporate inputs from model partitioning and approximation schemes to determine optimal configurations under varying operational conditions.

2. Energy Manager and Optimizer: A reinforcement learning-based monitoring system for application requirement, prioritization and computing load (cost) that provides feedback on energy consumption and system metrics (e.g., accuracy), allowing for adjustments to computation strategies to maximize efficiency without sacrificing accuracy. This practical consideration ensures that the vehicle-edge computing environment remains efficient and effective, adapting to application memory and processing conditions.

E) Prototype Development and Evaluation: This artefact includes the development of a prototype that embodies the above strategies for instantiating and testing the framework using:

1.Prototype Development: Constructs a functional system that integrates the adaptive model partitioning, approximation schemes, and the tradeoff framework.

2.Evaluation Metrics: Utilizes a set of metrics to assess energy efficiency, accuracy, and reliability under various operational conditions, providing feedback to improve energy-efficiency.

2.3. Research Method

This section outlines the research methods used to address the problem statement and research questions. To address the first question: What are the requirements to facilitate energy efficiency in data-intensive vehicular services?, a systematic literature review (SLR) was carried out using the guidelines discussed in the framework of Kitchenham and Charters [68]. Here in the question, 'requirement' refers to a theoretical or functional need that a particular design, component, or process must be able to perform. This is generally a specification or capability needed by a user to solve a problem or achieve an objective in an application [64]. In the scope of this thesis, requirements for energy-efficient computing of vehicular services are the specific conditions or capabilities that must be met or possessed by the system to enhance its energy efficiency in processing large volumes of data. The steps involved for identifying requirements using the literature review are as follows:

1. **Objective Definition:** Specify the goals of the literature review, focusing on identifying essential conditions for implementing energy efficiency in vehicular services.
2. **Search Strategy:** Create a search strategy which includes appropriate keywords, Boolean logic, and the selection of databases to ensure thorough literature coverage.
3. **Literature Search:** Execute a structured search strategy to locate pertinent studies, including both direct and indirect sources.
4. **Inclusion and Exclusion Criteria:** Evaluate studies for relevance based on a pre-established set of inclusion and exclusion criteria, removing literature that does not meet these benchmarks.

5. **Article Analysis and Categorization:** Breakdown and sort the relevant articles according to their contribution to understanding approximation techniques, the application of autonomous driving, and the role of edge computing.

6. **State of the Art Assessment:** Combine the insights from the literature to build an understanding of the current research landscape and pinpoint specific requirements for energy efficiency.

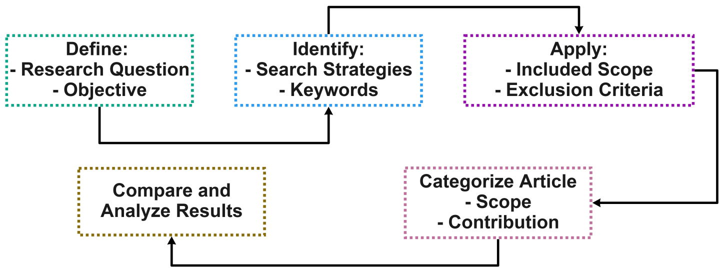


Figure 2.2: Literature review methodology based on Kitchenham and Charters [68].

For the second research question, 'Which foundational elements can enable approximation techniques on collaborative tasks?', we apply a combination of *case study analysis* and *comparative evaluation* as discussed by Yin [69] in "Case Study Research: Design and Methods". We recognize the value of case studies for their in-depth analysis of complex phenomena within specific contexts. In this research, data was collected through a series of controlled testbed experiments (in-lab setting) in which the proposed algorithms were implemented and tested under various deployment settings, including edge and server. These experiments involved running partitioning and approximation methods on different computing devices and network topologies, with resource usage, latency, accuracy, and energy consumption carefully monitored at each step. Experimental data is gathered by varying key parameters and configurations, enabling a systematic comparison of algorithm behavior across a range of practical scenarios. By extending these individual case studies through comparative analysis, both theoretical and practical understanding is improved by identifying common patterns and variations across the tested scenarios. This approach allows for a deeper exploration of foundational elements that support approximation techniques in collaborative computing tasks. Comparative evaluation enables a structured examination of these elements, strengthening the reliability and relevance of the research findings.

1. **Case Studies**: Examine real-world applications to gain insights into the practical use of approximation techniques in collaborative settings.

2. **Comparative Analysis**: Evaluate the effectiveness and challenges across various case studies to pinpoint successful strategies and common obstacles.

3. **Identify Case Studies**: Search experimental studies, white papers, consortium focus and proof-of-concept using digital databases like IEEE Xplore, Scopus, and

Google Scholar. Focus on keywords related to vehicle data services, connected vehicular services and applications, energy consumption, approximate techniques, model compression, edge AI and collaborative tasks.

4. **Detailed Analysis of Each Case Study**: For each selected study, document:
    - Specifications and configurations of AI models.
    - Details of collaborative tasks and their requirements.
    - Techniques of approximation used and details of their implementation.
    - Challenges encountered and solutions applied.
    - Effects on performance and energy efficiency.
5. **Analysis of Insights**: Combine the findings to identify patterns and insights that could guide applying approximation techniques in various contexts.
6. **Foundational Elements**: From the analysis, pinpoint key elements such as adaptability of models, task priority, resource requirement, limitations of hardware, and architecture of software needed for effective approximation.
7. **Expected Results**: This analysis will highlight effective strategies and essential requirements for the use of approximation techniques in collaborative vehicular tasks, supporting the development of an adaptive framework designed for energy efficiency and improved performance

For the third research question, "How to integrate the building blocks into an energy-aware adaptive software framework?", the involved research method and steps are as follows:

1. **Problem Reiteration:** Revisit the challenge of incorporating key components (approximation techniques, model partitioning, resource allocation, and deployment on edge devices) into a coherent software framework for energy management in vehicular applications.
2. **Designing the Solution:** Outline a framework design that includes:
    - **Approximation Techniques:** Strategy for selecting and integrating appropriate methods like quantization and pruning within AI model components.
    - **Model Partitioning:** Dynamic distribution of AI models across edge networks, tailored to energy limits and latency requirements.
    - **Resource Allocation:** Design algorithms to distribute computational resources effectively among tasks and devices.
    - **Model Deployment:** Develop protocols for scalable and secure AI deployment across distributed edge environments.
3. **Prototype Development:** Develop a framework prototype which includes the components described in the 'Design the Solution' while addressing challenges and impact which may include trade-offs between efficiency and energy consumption. The prototype in this thesis refers to framework deployed on testbed which includes heterogeneous edge devices.

4. **Performance Evaluation:** Assess the prototype on various model performance and system metrics including:
   - Energy usage
   - Computational efficiency
   - Accuracy of models
   - Scalability and integration with vehicular technologies
5. **Iteration and Enhancement:** Based on test results and evaluation, refine the framework. Adjust component integration, enhance adaptation strategies, and fine-tune resource management.

For the fourth research question, "What are the effects of implementing this framework on the accuracy of vehicular services and power consumption?", the involved research method and steps are as follows:

1. **Objective of Experiments:** Structured experiments are necessary to evaluate the framework's influence on service accuracy and power consumption.
2. **Comparative Examination:** Evaluate this framework against standard or other available solutions to discern its relative performance.
3. **Defining Performance Indicators:**
   - Accuracy Metrics: Precision, recall, F1-score.
   - Energy Metrics: Total energy usage, energy per operation.
4. **Experiment Setup:** Formulate a range of tests to determine the framework's behaviour under various conditions, such as distinct workloads and network setups. In the thesis, the experiment refers to the training and inference evaluation of AI models to capture energy and performance metrics in varied conditions, such as diverse computing and memory loads with varied data volumes to be processed.
5. **Data Gathering:** Compile data concerning the frameworks accuracy and energy metrics.
6. **Data Examination and Comparison:**
   - Assess the collected data to judge the frameworks impact on service accuracy and energy efficiency.
   - Benchmark against other models to establish this frameworks advantages or shortcomings.
7. **Analyzing Trade-offs:** Consider the balance between accuracy and energy usage to pinpoint possible refinements or enhancements.
8. **Framework Optimization:** Adjust the framework based on test and evaluation to enhance its efficiency.

9. **Additional Methodological Considerations:** As development and deployment include iteration and enhancement according to new components addition. The other considerations are:
    - **Benchmarks:** Utilizing new datasets for vehicular applications to ensure valid and robust comparisons.
    - **Real Conditions Testing:** Apply the framework on datasets that use real or near-real simulated environments to validate its effectiveness in practical scenarios.
    - **Statistical Methods:** Use statistical techniques to solidify the analysis of performance data.
    - **Sensitivity Tests:** Analyse how varying parameters influence the framework's efficiency.

2.4. Guiding Overview of Baseline Framework

The study of existing vehicular frameworks provides an understanding of the data processing, computing, and communication approaches, which is essential for collecting the requirements for an adaptive energy-aware framework. The currently proposed frameworks [70–72], address the challenges of memory and computation availability, communication in connected vehicles, and cloud and edge computing approaches to enable connected vehicular services. This section gives a brief overview of edge frameworks [70–72] and related architectures. These frameworks addressed the challenges of implementing compute-intensive tasks on resource-constrained embedded devices such as Nvidia Jetson TX1. Generally, these frameworks are integrated with Ubuntu OS using ROS or specifically designed custom OS (such as pi-os [73]). From the software perspective, the contributions made in the architecture of such frameworks can be summarized as layers, which mainly consist of service classification, scheduler, runtime framework, and communication layer (edge-server coordinator). The service classification layer helps in the hierarchical identification of tasks depending on the on-board computing operations and required computing resources. The second layer is a scheduler, which helps process the incoming data using scheduling algorithms. The data services integrator serves as a pipeline for the data after receiving it from the scheduler to be processed further on the hardware using the computing systems: GPU, CPU, video, and audio accelerator. The runtime framework layer acts as an orchestration component within the framework. The communication layer (vehicle-edge coordinator) enables vehicular communication by performing data offloading strategies.

Framework Overview (Guideline Figure): An overview of the multi-tiered software framework (represented in Figure 2.3) proposed in Lopecs [74] for edge AI deployment in autonomous vehicles is covered here. The Lopecs framework also serves as a baseline for this thesis. The Lopecs framework includes high-level classification of connected vehicle services (Quality-of-experience oriented), communication interfaces, and software components, including the operating system, scheduler, programming interface, runtime engine, and hardware resource composition. The following text describes the

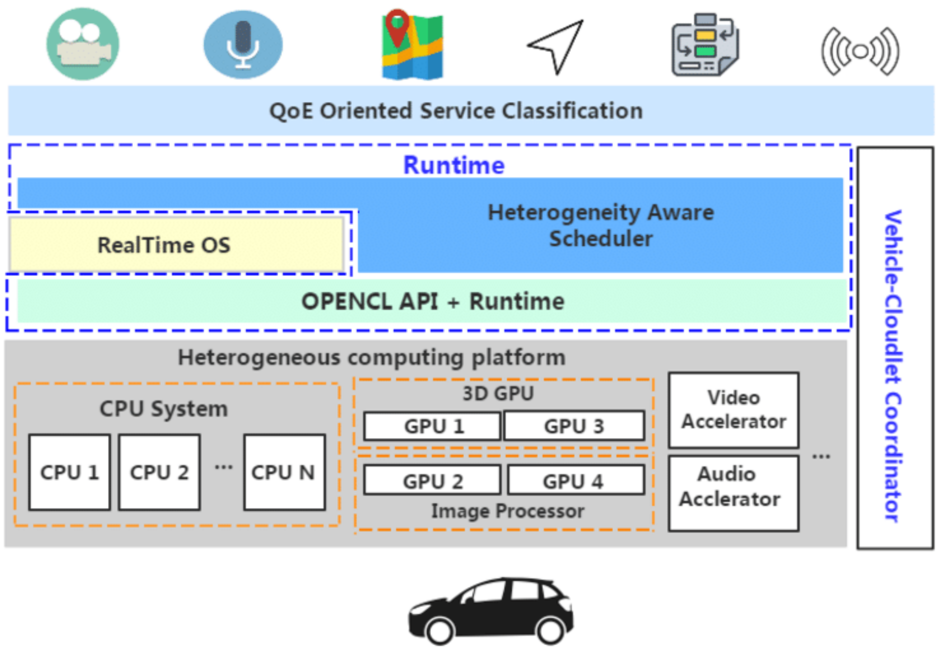


Figure 2.3: Lopecs: An energy-efficient framework for vehicular computing [74].

specific functionalities of these components and discusses how they interconnect to form an energy-efficient system.

1. QoE Oriented Services Profiler (Classification) At the first stage of the framework, Lopecs classifies the services integral to the autonomous vehicle ecosystem. These services may include:

- **Navigation Systems**: These systems are responsible for processing real-time geographic and traffic data to deliver dynamic routing and mapping information to the vehicle.
- **Multimedia**: This service provides entertainment and information to passengers through audio and visual media, enhancing the in-vehicle experience.
- **Voice Control**: By integrating advanced voice recognition and processing capabilities, this service allows for hands-free operation and interaction with the vehicle's systems, promoting safety and convenience.
- **Connectivity**: To enable communication with external networks, including other vehicles, roadside infrastructure, and cloud services, often referred to as Vehicle-to-Everything (V2X) communications.
- **Emergency Services**: Critical for the safety of passengers, this service ensures the vehicle can respond to and communicate during emergency situations effectively.

2.1 Runtime - Heterogeneity Aware Services Scheduler: The Scheduler's primary role is to oversee and manage the execution timeline of multiple concurrent tasks. It ensures:

- Optimized CPU and GPU usage by assigning priority to critical tasks.
- Reduced latency in task execution, crucial for real-time processing requirements of autonomous vehicles.
- Prevention of task collision and deadlock, maintaining system integrity and reliability.

2.2 OPENCL API: The Runtime Engine is the dynamic execution space where software comes to life. It is adept at:

- Managing active application processes, distributing them across CPUs and GPUs as needed.
- Handling runtime memory allocation efficiently, vital for performance-critical operations.
- Serving as the immediate executor of compiled code, interfacing directly with the programming interface and underlying OS.

3. Real-time Operating System (OS): The Operating System is the framework's base upon which all software is built. It is engineered to:

- Efficiently manage and allocate the vehicle's hardware resources.
- Provide a stable and secure environment for the execution of various software applications.
- Act as a fundamental intermediary, facilitating communication between the application layers and the physical hardware.

4. Heterogeneous Computing Platform: This layer combines the physical computational elements that constitute the vehicle's processing power, including:

- A series of Central Processing Units (CPUs) that provide the computational muscle for general task execution and multitasking capabilities.
- Graphics Processing Units (GPUs) specialized for parallel processing of visual data, a necessity for interpreting sensor inputs and environmental recognition.
- Dedicated AI accelerators are designed to efficiently handle specific tasks like video and audio processing, thus offloading these computationally intensive tasks from the CPUs and GPUs.

5. Vehicle-Cloudlet Coordinator: The communication interface is the gateway for bidirectional data flow between the autonomous vehicle and the external world. It facilitates:

- Real-time data transmission to and from edge computing resources.
- Integration with cloud services for data analytics, software updates, and additional computational support.
- The synchronisation of vehicle systems with up-to-date traffic and environmental data ensures optimal navigation and safety protocols.

3
Background: Theories and Practices

The increasing complexity of AI models used in autonomous vehicles has significantly elevated computational demands, raising concerns about energy consumption and operational efficiency. The widespread adoption of deep learning applications has driven the need for high-performance hardware, such as GPUs, which can consume substantial amounts of power. For instance, an electric vehicle with a 100 kWh battery could experience a 3–5% battery capacity drain in just hours when operating simple CNN models for object detection tasks [40, 75]. While cloud computing offers scalable and flexible computing resources, it often introduces latency and bandwidth constraints, where real-time decision-making is imperative for operational efficiency. The long communication distances between edge devices and cloud servers can result in unacceptable delays for time-sensitive applications like autonomous driving. Moreover, relying solely on cloud computing can increase network traffic and privacy concerns. Edge AI and approximate computing have emerged as promising solutions to address these challenges. Edge AI enables data to be processed closer to the source, reducing latency and improving responsiveness. Approximate computing techniques, which trade off some computational accuracy for significant energy savings, can further enhance the efficiency of AI models deployed at the edge. By combining edge AI and approximate computing, we can create more sustainable and efficient autonomous driving systems.

In this chapter, we sequentially discuss approximate computing, probabilistic and deterministic methods, edge AI technologies, model generalization and publicly available driving datasets. Also, by exploring fundamentals within the scope of approximate edge AI and connected autonomous, we answer research question 1, which focuses on requirements to facilitate energy efficiency in data-intensive vehicular services.

3.1. Approximate Computing (AxC)

Developing applications that use minimal energy requires optimization across several layers. At the **device level**, embedded platforms typically operate with limited power budgets, so their computations must be arranged to keep energy use low. At the **communication level**, sending data over networks consumes power, so a balance is needed between processing data locally and offloading it elsewhere. At the **cloud level**, hardware accelerators designed for particular tasks can reduce the energy required during large-scale processing. Approximate Computing (AxC) has gained renewed interest in recent years because it can address energy concerns at all three levels. AxC focuses on lowering energy usage, communication overhead, or circuit area by allowing some reduction in numerical accuracy or perfect correctness. Instead of aiming for full precision at all times, AxC uses the fact that many applications can tolerate a certain amount of error. These systems continue to operate effectively even when computations are not exact because the applications themselves have inherent error tolerance [27, 76–78]. This error-tolerance can come from:

- **Redundancy in large datasets**, which can hide small errors introduced by approximation.
- **Iterative refinement**, where later stages of an algorithm can correct earlier variations.
- **Acceptable value ranges**, where the output does not need to be exact to be useful.

These characteristics appear in areas such as image and video processing, localization, data analytics, and machine learning. In machine learning, for example, approximation techniques such as binarized weights can reduce computation cost with minimal effect on the final results [27].

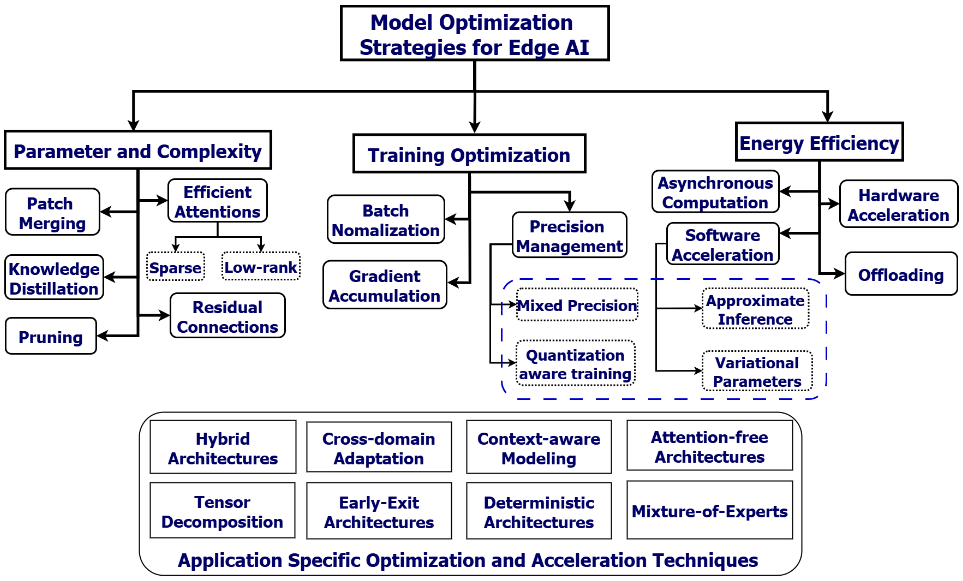


Figure 3.1: Approximation strategy categorization for ML models [43].

Approximation strategies for model optimization, as shown in Figure 3.1, are organized into three broad categories: parameter and complexity reduction, training optimization, and energy efficiency. Each category includes a range of techniques designed to balance computational demands, model accuracy, and resource constraints. For example, parameter and complexity strategies such as pruning, knowledge distillation, and patch merging reduce model size and operation count. Training optimizations focus on methods like batch normalization, mixed precision, and quantization-aware training to accelerate learning while controlling precision. Energy efficiency strategies integrate hardware acceleration, offloading, and approximate inference, targeting reduced power consumption on edge devices. Collectively, these approaches can be adapted and combined according to application-specific requirements, enabling efficient deployment of machine learning models in resource-constrained or real-time environments. The diagram also highlights advanced topics such as hybrid architectures and context-aware modeling, which further extend the range of application-specific optimization techniques for Edge AI.

3.2. Probabilistic and Deterministic AI Models

These models incorporate elements of probability theory to manage the uncertainties inherent in processing large amounts of data from varied sensing environments [34, 40]. The fundamentals are particularly adept at providing frameworks for reliable decision-making, even with incomplete or noisy data, providing an opportunity to estimate a balanced tradeoff. In energy-aware or sustainable computing, probabilistic and deterministic AI models can optimize performance while conservatively using computational resources [39, 52, 79]. These models can prioritize data processing tasks by calculating the probability of various outcomes, allowing the system to focus on more relevant or risky scenarios. This selective processing capability ensures that energy consumption is minimized without compromising the safety and reliability of the driving system. Furthermore, these models can adapt their computational accuracy based on the criticality of specific tasks, reducing unnecessary precision in less critical scenarios to save energy.

Several approaches are used to incorporate probabilistic reasoning into automated driving systems [20]. Bayesian networks are widely used to manage uncertainty in sensor measurements and environmental conditions by updating system beliefs when new data arrives [27, 80]. This makes them suitable for sensor fusion and short-horizon scene understanding in autonomous vehicles [20]. Markov Decision Processes (MDPs) support decision-making when outcomes are uncertain, allowing the system to weigh immediate actions against future effects. They are often used in planning and control, where the driving environment changes continuously and actions must adapt accordingly [80, 81]. Monte Carlo methods provide sampling-based estimation for tasks such as motion prediction, route selection, and risk evaluation. These methods produce outcomes with measurable uncertainty, helping the system choose actions that remain reliable across varying driving conditions [81].

The selection of these techniques in application shows their effectiveness and frequent application in both academic studies and practical systems, as referenced above. Collectively, these methods offer established frameworks for handling uncertainty, supporting adaptive control, and improving the reliability of perception and decision-making modules in connected and automated vehicles.

A neural network is referred to as probabilistic when uncertainty is incorporated into its weights. This classification includes a broader set of models than those strictly defined by the Bayesian framework. In the traditional Bayesian formulation, a prior distribution

$p(w|\phi)$ with hyperparameters ϕ is assigned over the weights $w \in \mathbb{R}^{D_w}$, and a likelihood $p(D|w)$ describes the observed data D. The main objective is to characterize the posterior distribution $p(w|D)$. During posterior inference, the hyperparameters ϕ of the prior are kept constant. Since a closed-form solution for the posterior is typically unattainable, approaches such as Markov Chain Monte Carlo (MCMC) or Variational Inference (VI) are adopted [80]. In VI, an approximate posterior $q(w)$ is optimized to maximize the Evidence Lower Bound (ELBO), given by $\mathbb{E}_{q(w)}[\log p(D|w)] - \mathrm{KL}(q(w)\|p(w|\phi))$.

An alternative methodology involves the introduction of noise into the weights, followed by marginalization, represented as:

$$\arg\max_{\phi} \log \int p(D|w)\, p(w|\phi)\, dw,$$

which is frequently referred to as Type-II maximum likelihood (ML) or empirical bayes. This method is distinct from the Bayesian approach in that the prior $p(w|\phi)$ is learned during the process. In contrast, the Bayesian approach maintains a fixed prior throughout. Previous research has shown that the prior can be learned while simultaneously performing posterior inference over the neural network weights using VI [80]. The strategy of maximizing marginal likelihood has been applied in multiple areas of machine learning, including evidential deep learning, prior networks, and PAC-based deep learning.

3.3. Taxonomy of Edge AI for CAV

In this section, we discuss the taxonomy adopted and used in this research. Initially, traditional AI approaches for autonomous driving are described. Following this, applications of Edge AI and computing are outlined. Next, approximation methods and compression strategies are defined. Finally, energy-aware mechanisms and requirements for vehicular environments are discussed. An overview of this categorization is provided in Figure 3.2. [1]

Development Methodology: The taxonomy is developed from the systematic literature review as described in our survey [20], which included an analysis of over 400 articles and technical sources covering AI models, model compression, edge and cloud computing, driving datasets for vision tasks (more than 80), and vehicular AI frameworks.

1. *Systematic Literature Collection:* We gathered a wide range of peer-reviewed publications (from databases such as IEEE Xplore, ACM, SpringerNature, Scopus) focusing on the latest research (from the past 5–7 years) in areas relevant to AI-enabled Connected and Autonomous Vehicles (CAVs), edge AI, approximate computing, and energy-aware mechanisms.

2. *Screening:* The collected works was filtered based on relevance to CAV applications and technical focus on model optimization, deployment, and energy efficiency. This process ensured the inclusion of foundational studies, major surveys, and leading-edge application papers.

[1] Katare, Dewant, et al. 'A survey on approximate edge AI for energy efficient autonomous driving services.' IEEE COMST (2023)

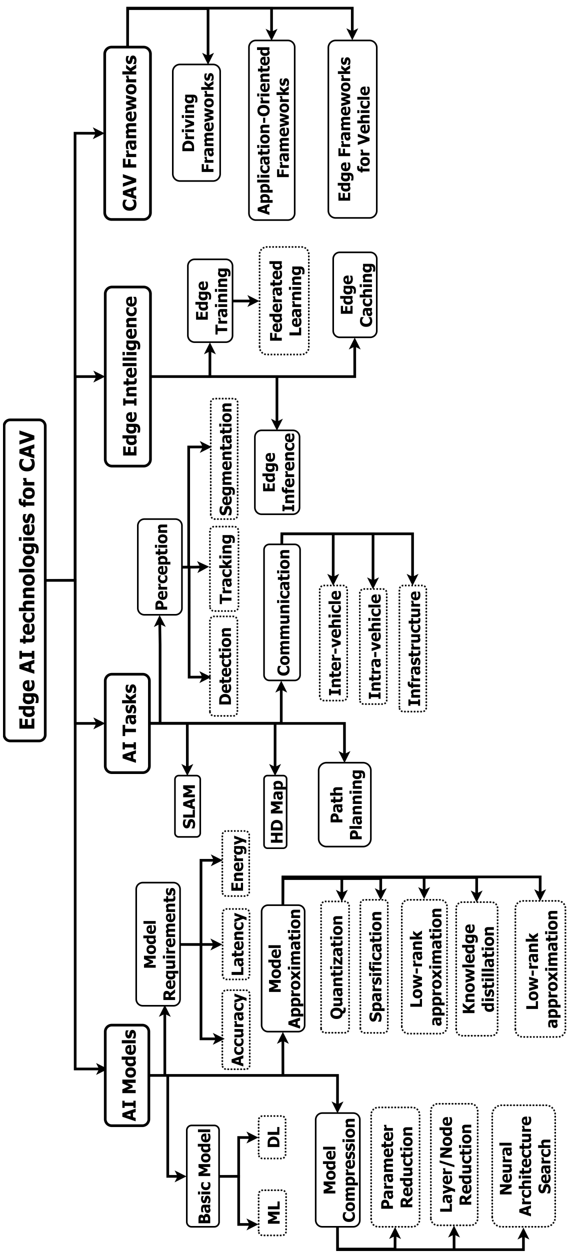


Figure 3.2: Taxonomy covered in this research (result of research question 1).

3. *Thematic and Comparative Analysis:* The literature was categorized according to themes such as: (i) AI models for CAVs, (ii) model compression and approximation

(e.g., quantization, pruning, knowledge distillation), (iii) edge/cloud system architectures, (iv) dataset diversity and evaluation, and (v) energy-aware orchestration.

4. *Iterative Refinement and Validation:* The taxonomy categories were refined through multiple rounds of analysis (as also shown in 2.2) and feedback from the peer-review process during survey publication.

5. *Synthesis and Structuring:* The final taxonomy is structured to reflect key topics: (1) AI models and perception for autonomous vehicles, (2) Edge AI and computing architectures, (3) Approximation and model compression strategies, and (4) Energy-aware mechanisms and deployment requirements in vehicular environments.

3.3.1. AI Models & Autonomous Vehicles

An autonomous vehicle is considered an independent system that navigates from source to destination by perceiving its environment through sensors and processing sensor data using intelligent algorithms. Advancements in CAVs have been facilitated by developments in vehicle sensors, embedded systems, and intelligent algorithms. These developments have enhanced connectivity, infotainment, electrification, and automation in vehicles. Perception sensors (such as camera, LiDAR, radar), positioning sensors (including GPS, GNSS), and communication modules are deployed to assist or replace the driver using AI models.

- **Basic Model:** AI models used for automating or supporting driving tasks are typically divided as follows:
  - *Machine Learning:* Supervised, unsupervised, and reinforcement learning techniques have been applied in autonomous driving.
  - *Deep Learning:* As a subset of machine learning, deep learning involves various neural network architectures trained to extract complex features from structured and unstructured data.
- **Model Requirements:** AI models are developed with specific requirements based on the driving tasks involved. For example, localization, emergency braking, and detection of obstacles or traffic signs require a high degree of accuracy. In this study, the following model requirements are considered:
  - *Accuracy:* The objective of utilizing AI models is to minimize human error in driving and to achieve the required accuracy for given driving tasks. Accuracy is quantified as the ratio of correct model predictions to the total number of predictions.
  - *Latency:* Execution time varies by driving task; detection and localization often require latencies of only a few milliseconds. In AI models, latency refers to the processing time for a specific application.
  - *Energy:* Achieving high accuracy and low latency typically requires powerful computation, resulting in increased energy consumption. Energy usage (in Joules) can be estimated from the power consumption (Watts) of the AI models during operation.

- **AI Model Compression:** Compression strategies enable the deployment of large models or datasets on resource-constrained devices. Both lossless and lossy methods have been explored for vehicular AI tasks. Common approaches include [82]:
  - *Parameter Reduction:* Model parameters are reduced to lower complexity and speed up training or inference. Removing non-contributory weights or layers (pruning) is a widely used method for compression.
  - *Layer/Node Reduction:* Reduction in the number of layers or nodes is often adopted to meet memory and computation constraints, helping to balance accuracy and resource usage. Examples include minimal matrix operations and parameter-sharing.
  - *Neural Architecture Search:* This approach searches for optimal network architectures or hyperparameters, with objectives such as downsizing models or addressing bandwidth constraints in vehicular communication.
- **Approximate Techniques:** Approximate computing uses methods from mathematics and engineering, such as probabilistic circuits, to balance metrics for faster computation at the cost of reduced precision [83–85]. Software and model compression strategies in CAV frameworks may be categorized under approximation methods. However, these techniques do not always address energy efficiency, particularly from computational and communication perspectives.
  - *Quantization:* In vehicular AI, quantization reduces the numerical precision (e.g., from 32-bit floating point to fewer bits), minimizing model size and communication bandwidth. This is inspired by discrete information storage in biological systems [86].
  - *Sparsification:* Here, vectors are represented by retaining only significant components, with other elements set to zero. Sparsification is often used in collaborative or distributed learning, such as federated learning, to reduce communication overhead.
  - *Low-Rank Approximation:* Reduced computation for AI models is achieved through methods such as Tucker or Canonical Polyadic decomposition in convolutional neural networks. While model size is minimized, accuracy can be affected.
  - *Knowledge Distillation:* Larger neural networks are approximated by compressed models, although maintaining performance in this compressed form remains challenging.

3.3.2. CAV (AI) Tasks

The scope of AI-enabled driving tasks includes perception, HD mapping, localization, path and motion planning, actuation, and vehicular communication. In this thesis, tasks are further differentiated based on data processing strategies, feature extraction methods, and hardware platforms.

- **Perception:** Perception tasks provide scene understanding and are carried out using sensors such as cameras or LiDAR on the vehicles onboard unit or through ecosystem devices (e.g., CCTV cameras). These tasks are typically implemented using CNN or DNN models running on GPUs, which involve significant computational and energy costs due to dense network layers.
- **SLAM:** Simultaneous localization and mapping (SLAM) enables vehicles to determine their position using sensor data. Models supporting SLAM are often memory and compute-intensive, and additional complexity arises from real-time inference demands.
- **HD Map:** Sometimes referred to as a 3D map, this feature provides a detailed three-dimensional representation of the vehicles environment and is used alongside detection and localization tasks.
- **Communication:** Communication in vehicular systems is dynamic and heterogeneous, occurring within the vehicle, between vehicles, and across infrastructure. Modern applications depend on evolving hardware, software, and sensor technology. Increased autonomy requires effective sharing of data, model weights, and algorithms across the network, which increases both data volume and bandwidth demand.
- **Path/Motion Planning:** Path and motion planning enable vehicles to navigate safely from source to destination. While algorithms such as A* have been traditionally used, modern approaches integrate vision-based AI models to predict paths and avoid obstacles.

3.3.3. Edge AI and Connected Vehicles

Cloud computing was initially proposed to manage computation and decision-making in connected vehicles [73, 87, 88]. However, reliance on cloud computing introduces challenges such as transmitting large volumes of data, risks to data privacy, potential data leakage, and vulnerability to adversarial or poisoning attacks [88]. To address these concerns, computation has been shifted closer to the data source through edge computing [89]. This method has been extended with the concept of edge intelligence, which enables AI applications to be deployed on edge devices near the data source. Edge AI can improve privacy and security and can address the challenges of distributed computation and communication in the connected vehicle ecosystem, supporting applications such as driver assistance, infotainment, decision-making, and safety-critical services [54, 90]. The concept is typically described within categories as edge based training, edge inference, and caching at the edge.

- **Edge Training:** With the dynamic and distributed nature of future vehicular systems, edge training supports collaborative or federated learning among devices, allowing for local model updates and re-training.
- **Edge Inference:** AI models are deployed on resource-constrained devices for inference at the edge. When deploying AI in connected vehicles, several considerations must be addressed:

  - *Latency:* Vehicular applications often have strict latency requirements. In connected vehicles, latency arises from sensor data processing, fusion, computation, and inter-device communication.
  - *Real-Time Inference:* Real-time inference is necessary for many applications. Processing data close to the source helps meet latency and timing requirements, but incurs significant computation and energy costs.
  - *Offloading:* Data and computational offloading to nearby edge servers helps alleviate resource constraints in low-power devices, reducing the amount of data sent to the cloud.
  - *Heterogeneity:* Diverse data types, device capabilities, communication protocols, and network architectures introduce deployment and management challenges in edge-cloud-vehicle systems.
  - *Reliability:* Edge AI supports low-latency and real-time applications, helps maintain data privacy, and can be used in safety-critical scenarios. Nonetheless, issues such as communication congestion, delay, and bandwidth limitation may arise in specific environments (e.g., rural or highway settings).

- **Edge Caching:** Frequent data exchange for model training or updating requires caching functions at the edge, which manage data collection, storage, processing, and real-time labeling in distributed environments.

3.4. Energy Efficient Edge Frameworks

This section gives a brief overview of existing energy-efficient edge frameworks [70–72] and related architecture. These frameworks addressed the challenges of implementing compute-intensive tasks on resource-constrained embedded devices such as Nvidia Jetson TX1. Generally, these frameworks are integrated with Ubuntu OS using robot operating system or specifically designed custom OS (pi-os [73]). Previous work in energy management includes software strategies such as data compression, reducing the model's complexity, and algorithms [91, 92] to optimize related components. Techniques like model pruning and efficient neural network designs have also helped to reduce the computational demands by using compressed models used in the perception, mapping, and navigation of connected autonomous vehicles [93–95].

While these methods have effectively reduced model and computational parameters, the compression scope does not consider dynamic vehicular environments' variable computational requirements and characteristics. As a result, energy-efficient strategies operating under static conditions often have less than optimal performance in unpredictable situations where the environmental and operational conditions are dynamic [96]. Existing frameworks such as LoPECS [74] discuss a low-power edge computing system to reduce power consumption in autonomous driving applications. The system uses a heterogeneous computing approach and dynamic task offloading to edge cloudlets, showing onboard energy savings for V2X applications. This framework uses a Heterogeneity-Aware Runtime Layer and a vehicle-edge Coordinator to optimize the user experience with extended battery life and enhanced performance of driving services.

VECMAN [97] introduces energy-aware resource management in vehicular edge computing systems by addressing the challenge of high energy consumption in collaborative applications through optimized resource-sharing and task-offloading strategies. The framework reduces computational energy consumption, using key components such as a resource selector algorithm and an energy manager algorithm, which enhance system reliability and energy efficiency. Using a deep deterministic policy gradient (DDPG) algorithm, a deep reinforcement learning-based framework is discussed for optimizing resource allocation in the Internet of Vehicles [98]. The framework reduces the mobile network operator's energy costs while ensuring timely task completion. This study shows the potential of DRL methods to achieve robust, real-time decision-making in complex vehicular network environments.

A carbon-aware framework for AIoT ecosystems focusing on sustainable computing practices is discussed in [99]. The framework discusses energy-efficient communication and low-carbon task offloading. The framework includes a multi-source model for communication and a carbon-aware multi-channel exploration offloading decision algorithm. Experiments show that it outperforms existing methods in reducing data acquisition errors, energy consumption, and carbon emissions, enhancing the overall sustainability of AIoT ecosystems.

3.5. AI Model Generalization and Driving Datasets

Generalization enables AI models to learn from previously unseen class samples, facilitating robust performance across diverse scenarios. It is typically assumed that the classes in the training and testing distributions are similar; however, constructing test sets that capture all possible distributions is resource-intensive. Therefore, neural networks and AI models are trained to promote generalization, which reduces the influence of dataset bias and supports practical deployments [100]. Over recent years, strategies have been proposed to address dataset bias, including transfer learning, covariate shift adaptation, domain adaptation, adversarial techniques, and out-of-distribution generalization [100]. Hardt et al. [101] introduced a fairness metric based on demographic parity for supervised learning, supporting the evaluation of model fairness during both training and post-processing. Such approaches can also function as privacy-preserving measures in AI applications.

Bias in AI models refers to systematic misclassification or prediction for certain classes, often resulting from inappropriate modeling choices. For example, a biased estimator may be used to minimize variance for small datasets, inadvertently introducing bias for future cases [102]. Bias can also arise when overlaying irrelevant data during model training or neglecting significant ground truth features that are essential for regression or prediction. A specific case is a forward collision warning system utilizing radar or camera data, where features unrelated to separation distance, acceleration, or velocity may be inadvertently incorporated [103]. This type of scenario can be investigated with datasets featuring object detection and tracking, utilizing both camera and radar information to assess static and dynamic vehicle behaviors. In neural networks, high bias may result from missing connections between input features and output predictions (underfitting), while high variance is evidenced by strong training set performance but poor generalization to new data (overfitting) [104]. Addressing these issues requires balanced datasets with equal class representation and monitoring of neural activity across layers during training [105].

Dataset bias remains a persistent challenge for AI in real-world applications, such as autonomous driving [106]. Sources of bias include uneven class distributions, presence of out-of-distribution samples, limited data under varying conditions (weather, lighting), label inaccuracies, and inconsistent viewpoints [107]. Annotation bias, described in [108], can further amplify both algorithmic and model bias. To mitigate these biases, strategies such as re-weighting schemes can be applied during DNN training [109]. This approach involves pre-processing the dataset to prioritize unbiased or underrepresented labels. Robinson et al. [110] developed a balanced dataset with subgroup-specific thresholds to assess performance across data subgroups, highlighting the influence of dataset composition on verification outcomes. Cui et al. [111] proposed a Bayesian model to improve dataset annotation quality using MAP inference, refining data labeling at both the object and frame level.

Detecting and mitigating bias in AI models performing object detection and classification with imbalanced data is challenging, as certain object classes may dominate feature learning and recognition. Selecting neural network architectures that generalize well and exhibit minimal bias, even when trained with imbalanced datasets, is critical. To address these challenges, this study investigates bias present in widely used AI models, including those trained on established datasets, models adapted via transfer learning, or cross-validated using driving datasets. The datasets considered include biased car datasets [107], nuScenes [112], and the Waymo dataset [113], all of which feature classes with overlapping attributes. To relate algorithmic and system performance to robustness and energy efficiency from the perspectives of model learning and data representation (see chapter 4), the following objectives are addressed:

- AI models for object classification and detection are examined, with a focus on the relationship between accuracy and data diversity.
- Metrics such as selectivity score, accuracy, and minimum average precision are evaluated across varying data diversity.
- Bias metrics are analyzed and compared with model performance parameters using cross-validation and transfer learning experiments.

3.6. Correlations of Metrics:

This research also explores metrics correlation to analyze algorithmic performance, algorithmic robustness, system operations, and energy efficiency within connected vehicle services. An overview of the metrics study is shown in the correlation matrix figure 3.3, describing tradeoffs between algorithmic accuracy (accuracy, mAP, IoU) and system performance metrics (latency, MACs), particularly noting that higher accuracy in object detection tends to increase processing time, potentially impacting real-time functionality.

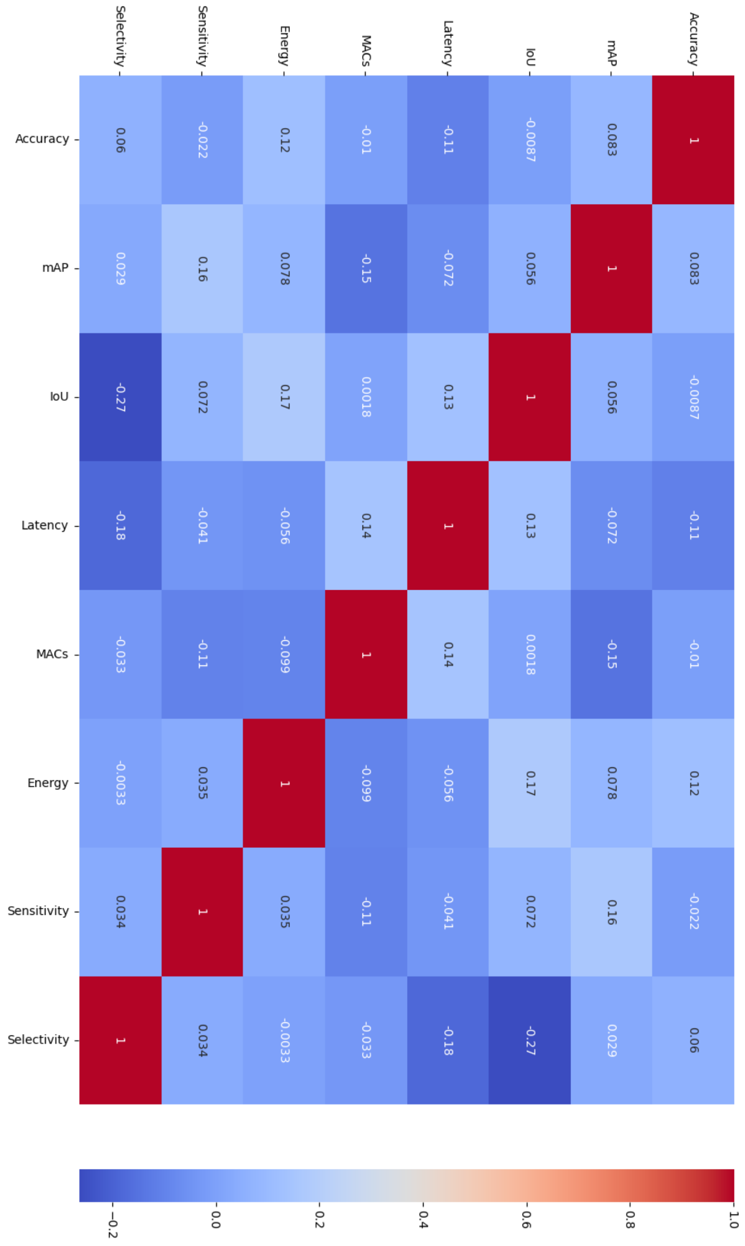


Figure 3.3: Metrics correlation (algorithm, system, energy, and robustness) profiled for ResNet34 on Jetson Xavier NX.

Furthermore, robustness metrics such as sensitivity and selectivity entail complex interactions with traditional performance metrics, emphasizing the challenges in optimizing neural networks for performance and robust decision-making under dynamic conditions. Energy considerations are also critical, as shown by the relationships between energy usage, MACs, and Latency, underscoring the impact of computational demands on power consumption. These insights are important for refining AI models to balance accuracy, processing speed, and energy constraints, essential for advancing autonomous vehicle technologies. Notably, this thesis is the first to explore such correlations amongst the common (algorithmic performance, system) and overlooked metrics (robustness, energy). Understanding these relationships enables system designers to prioritize model and hardware optimizations for specific application constraints. Future research should focus on algorithm optimization to mitigate accuracy-latency trade-offs and explore energy-efficient computing architectures to enhance system performance without compromising on energy usage. Additionally, enhancing model robustness in variable driving conditions remains a critical area for development, potentially through models that adjust computational strategies in real-time. This research highlights the complex interplay between various performance metrics in AI systems, with significant implications for the design and operation of more reliable, efficient, and robust automated driving technologies.

3.7. Design of Vehicle-Edge Processing Pipeline

The need for an Edge AI training and inference pipeline, capable of handling large data volumes and processing them through a neural network for decision-making tasks based on task prioritization, was identified while exploring Research Question 1. An overview of the proposed Edge AI processing pipeline for future connected vehicular services is shown in Figure 3.4, where on-board computing and processing of the data at the edge is discussed. Four major components are included in the proposed AI processing pipeline. Data from the vehicle's surroundings is captured by sensing units, present in the vehicle (camera, lidar, radar, GPS, the communication units, including on-board and cellular connectivity), as the first component.

The second component involves computation and decision-making performed by an edge device situated in the vehicle. Data is processed by this device using deep neural networks, facilitating driving tasks such as perception, SLAM, and vehicular communication. Considerable on-board energy is required for the complex computation and decision-making stages, making it essential to consider energy-efficient strategies within the edge intelligence paradigm. To address this, the computation and decision-making segment is further divided into a data processing pipeline and a computing unit. Tasks such as offloading, data labeling, real-time compression, legacy data management, and sharing of processed data with other entities (e.g., nearby vehicles or edge servers) are handled within the data processing pipeline. The critical concerns of memory and power in resource-constrained edge embedded devices can be mitigated through these processing strategies. The computing unit processes refined data using deep neural networks to generate model weights for driving-related applications. Additional acceleration and approximation methods, including neural network compression, data fusion, and early-exit network designs, may be incorporated to optimize the neural network computation.

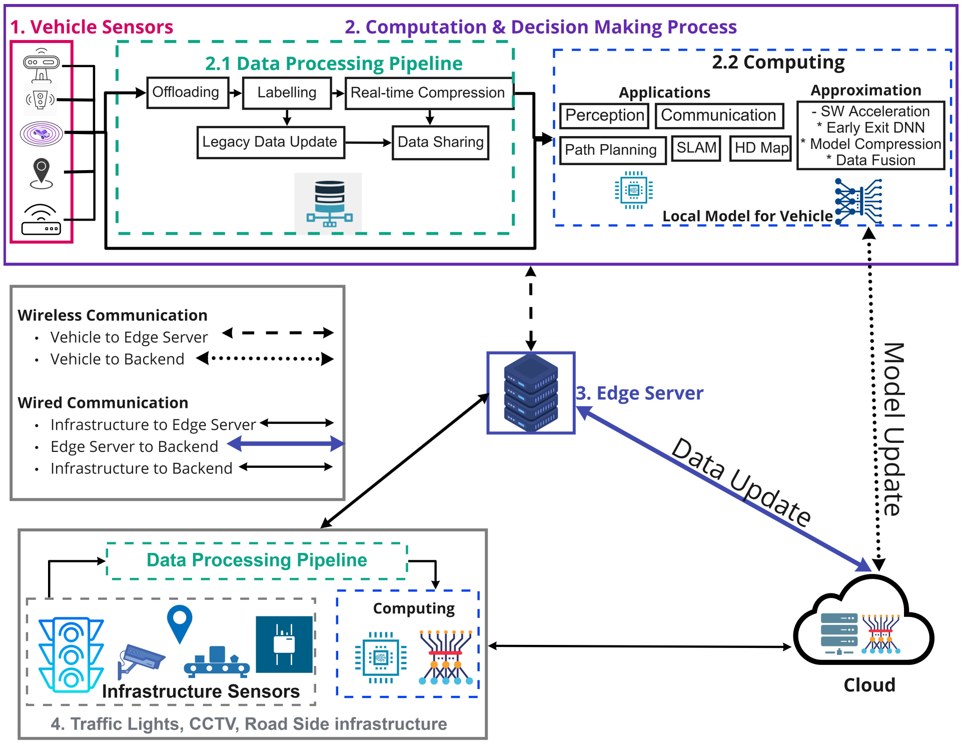


Figure 3.4: A visual representation of the Edge-supported autonomous driving system, which include data acquisition from vehicle-mounted sensors, the computational and decisional block, the Edge server, Infrastructure sensor inputs, and the cloud's repository of autonomous vehicle data with global deep neural network model. (Figure adapted from [20]).

The third component consists of an edge server that manages large-scale data processing and enables communication across the vehicular ecosystem. Communication is facilitated in several directions: vehicle to edge-server (for raw data transfer), edge-server to vehicle (for DNN weights and refined or processed data), edge-server to infrastructure, and edge-server to backend cloud. The edge server is responsible for handling computationally intensive tasks, implementing lossless compression, optimization, and software-level approximation approaches to reduce the on-board energy requirements of autonomous vehicles, thereby supporting end-to-end energy efficiency.

The fourth component comprises roadside infrastructure, which integrates a sensor suite (e.g., CCTV, traffic signals, lidar, communication modules, and GPS), similar to the sensing capabilities within the vehicle. This infrastructure supports operations such as intelligent traffic flow management, real-time monitoring, and map updating. As depicted in Figure 3.4, the infrastructure component also contains a data processing pipeline, responsible for tasks including offloading, data labeling, real-time compression, and the exchange of data or model updates over wired connections to the edge server and backend cloud. When model or data updates are necessary, the backend cloud communicates with the vehicle, edge server, and infrastructure sensors. Model weights and updates should be distributed among the backend cloud, vehicle, and edge server over both wireless and wired networks, thus promoting accuracy and supporting collaborative driving applications.

Overall, the vehicle-edge processing pipeline provides an initial knowledge base to develop efficient data processing mechanisms, software-level and model acceleration techniques such as model partitioning, approximate computing mechanisms, and adaptive deployment strategies by prioritizing vehicle services (e.g. HD map, Perception) on the basis of model performance, application requirements, and system-level metrics.

3.8. Conclusion

This chapter serves as the foundation for understanding the overall taxonomies that enable the development of energy-efficient AI systems in the context of CAV services. Using the background research, taxonomy formulation and literature review, this chapter addressed the requirement research question (RQ 1) by identifying key components required for developing an energy-aware adaptive framework, thereby fulfilling the requirements for enhancing energy efficiency in data-intensive connected vehicular services. From the exploration of approximate computing, which provides a method to reduce energy consumption, to the discussion of probabilistic and deterministic models that enhance decision-making under uncertainty, to model partitions and deployment mechanisms, the overall functional requirement for advancing autonomous vehicle technologies is covered.

Based on the literature and system motivations discussed, three components emerge as central to enabling energy-aware operation in vehicleedgecloud computing environments:

1. **Approximate computing mechanisms**, which allow controlled reductions in precision to lower computation cost while keeping output quality at usable levels.

2. **Model partition strategies**, which divide models across vehicle, edge, and cloud hardware to match available compute capacity and energy budgets.

3. **Resource allocation methods**, which select where and how each model component should run based on power limits, workload demand, and communication conditions.

These components are supported by evidence presented throughout this chapter. Modern CAV systems rely on continuous perception and planning workloads that require frequent processing of high-resolution sensor streams. Running these workloads at full precision increases computation and power draw, which is directly tied to battery usage in hybrid and electric vehicles. Studies indicate that the computing hardware in autonomous vehicles can consume from tens of watts to several hundred watts or more for real-time perception alone. This added demand reduces driving range and contributes to operational energy cost. Therefore, reducing computation where acceptable, distributing tasks across available hardware, and allocating resources adaptively are necessary to maintain system performance while lowering onboard energy use.

Furthermore, the exploration of Edge AI shows how computational tasks can be efficiently managed closer to data sources, thereby minimizing latency and reducing communication and cloud computing costs, which are key factors in energy consumption for collaborative applications, including connected autonomous vehicles. The insights into AI model generalization and the challenges of dataset biases highlight the importance of robust model training to ensure fairness and accuracy, which is critical for the acceptance and reliability of autonomous systems. The concepts and strategies discussed here are high-level components which can be integrated to develop an energy-aware adaptive framework for a vehicle-edge computing environment. The subsequent chapters will build on this knowledge to present a detailed analysis of specific implementations, their evaluations, and the practical implications of deploying these advanced technologies in real-world scenarios.

4 BiasDet: Biases Detection in Datasets and Algorithms

The fairness of machine learning models used for perception applications in autonomous systems should be considered as a priority due to the risks present in urban driving environments, especially for the vulnerable road users, which include classes such as pedestrians, cyclists, and motorcyclists. The majority of research in the perception domain specifically focuses on improving the accuracy measure of models over a given dataset by enhancing class performance metrics; which generally overlooks the hidden pattern of bias inheritance in the AI models, class imbalances, and disparities within the vision datasets. Before exploring the accuracy, energy and performance metrics optimization strategies, in this chapter, we focus on understanding model learning representations and dataset impacts and related issues by investigating class imbalances among vulnerable road users, focusing on analyzing class distribution, evaluating metric performance, and assessing bias impact. Utilizing popular vision datasets (automotive), CNN models and Vision Transformers (ViTs), we understand and evaluate detection disparities of underrepresented classes.

This chapter contributes to developing reliable models while enhancing inclusiveness for minority classes in datasets. This exploration directly addresses research question 2, which is described as: which components can enable task deployments energy-efficiently and collaboratively in vehicle-edge-cloud computing?. By identifying biases as foundational challenges within AI models, this chapter explores how different AI models can be optimized for energy saving and if a correlation exists between model performance and model behavioural metrics. The chapter is based on the publication addressing bias analysis and mitigation strategies in datasets. [1]

The methodologies used here include a detailed analysis of driving datasets, focusing on class distribution and the evaluation of bias impact. This approach highlights the disparities and assesses the effectiveness of various mitigation strategies. The structure of this chapter is as follows. Section analysis of driving datasets covers two major datasets (nuScenes [112] and Waymo [113]), their class distributions, and the inherent biases affecting AI model performance. It helps in understanding how biases in data affect overall model accuracy and fairness. The following section, learning representation discusses statistical representation of classes, exploration of behavioural metrics to under-

[1] Work published as: Katare, Dewant, et al. Analyzing and Mitigating Bias for Vulnerable Road Users by Addressing Class Imbalance in Datasets: IEEE Open Journal of Intelligent Transportation Systems (2025).

stand models learning processes, undersampling and oversampling issues, and various approaches for data and sample inclusion such as data augmentation, resampling, and applying metric-specific learning for models. Evaluation section discusses vulnerable classes which covers implementing these strategies, analyzing model learning representation, and improving accuracy. The last section conclusion reflects on the implications of these findings for deploying AI in autonomous vehicles, emphasizing the need for statistical improvement in the dataset for the model training, validation and fine tuning.

4.1. Analysis of Driving Datasets

The deployment of autonomous vehicles (AVs) have been proposed as an improvements to existing traffic management and driving safety conditions. However, the effectiveness of such technologies depends primarily on the accuracy and impartiality of perception systems [114, 115]. These systems, especially those designed for object detection, play a central role in identifying road users, including pedestrians, cyclists, motorcyclists, and mobility-impaired individuals [40, 116, 117]. Recent research has documented the presence of biases within vision datasets, including those used in autonomous driving, which may result in unequal treatment of certain categories of road users [118, 119].

This study covers two widely used datasets, nuScenes [112] and Waymo [113], and investigates class imbalance and the potential for bias transfer when models are trained on data containing class disparities. The chapter presents an analysis of the representation of vulnerable classes and evaluates object detection model accuracy and fairness using class distribution assessments, model performance analysis, and bias impact studies. Class imbalance refers to an unequal distribution of categories within a dataset [120, 121]. In autonomous driving datasets, this often results in a predominance of certain classes, such as vehicles, while categories such as pedestrians, cyclists, motorcyclists, and particularly mobility-impaired individuals are less frequently represented [118]. This skew can produce perception systems that are less proficient at identifying less common, yet important, classes [118, 122, 123].

Previous research suggests that such imbalances lead to the propagation of bias from the dataset into AI models, causing these models to prioritize the detection of majority classes [124]. Figure 4.1 presents a heatmap from model tests, where input images are depicted on the left and class-specific gradient flow is visualized on the right. The impact of these dataset imbalances has been highlighted by real-world incidents, such as the Uber self-driving car crash [125]. The necessity for datasets that are balanced, diverse, and representative of a broad spectrum of real-world situations is therefore underscored.

Within this context, the chapter investigates algorithmic performance and model robustness as foundational aspects for enabling approximation techniques in collaborative vehicle-edge tasks. The research gaps considered in this chapter include:

- In what ways do class representation and disparities within datasets affect the accuracy of AI models (research question 2.1)?

- How do biases in AI systems impact the performance and accuracy of applications in collaborative and distributed edge AI environment (research question 2.2)?

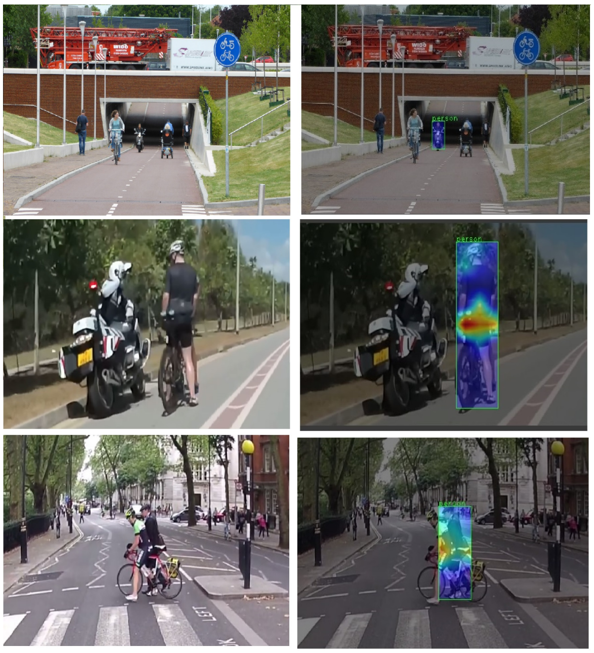


Figure 4.1: Three samples that show testing results when a single class is predominantly present during model training.

Compared to previous works, which are dependent on the analysis of F-1 score, false positives(FP) and false negatives(FN) for bias impact assessment tasks, our research explores sensitivity and selectivity score for NNs and layer-wise relevance propagation for vision transformer models within a framework for class-specific learning/evaluation process to identify bias inheritance. The framework then includes bias assessment using FP, FN, and model re-calibration process. This is to ensure the model is fair and works well when retrained with new classes or sub-samples. We also identify key shortcomings in current vision datasets for autonomous driving. This is done through metric-specific learning, aiming to create more fair and reliable systems. The aim is to understand model learning representations over a dataset and contribute to generalizing AI algorithms over unseen or underrepresented classes in the datasets. [2]

4.2. Learning Representations

When a dataset D contains an uneven distribution of classes $\{C_1, C_2, \ldots, C_n\}$, the training of machine learning model such as neural networks and vision transformers (ViTs) may result in the inheritance of biases linked to class imbalance. Such biases are often attributable to over-represented normal driving conditions and the relative scarcity of challen-

[2]Work published as: Katare, Dewant, et al. Bias detection and generalization in AI algorithms on edge for autonomous driving. 2022 IEEE/ACM 7th Symposium on Edge Computing (SEC). IEEE, 2022.

ging scenarios within the dataset. In neural networks, these biases tend to be observed in neurons' sensitivity and selectivity scores, while in vision transformers, attention allocation patterns may reveal bias. Accordingly, a detailed methodology is proposed that employs statistical techniques for identifying and mitigating biases, incorporating dataset analysis, bias impact assessment, mitigation approaches, metric behaviour analysis, and model recalibration, as illustrated in Figure 4.2. The subsequent sections detail the methodology for model training and evaluation.

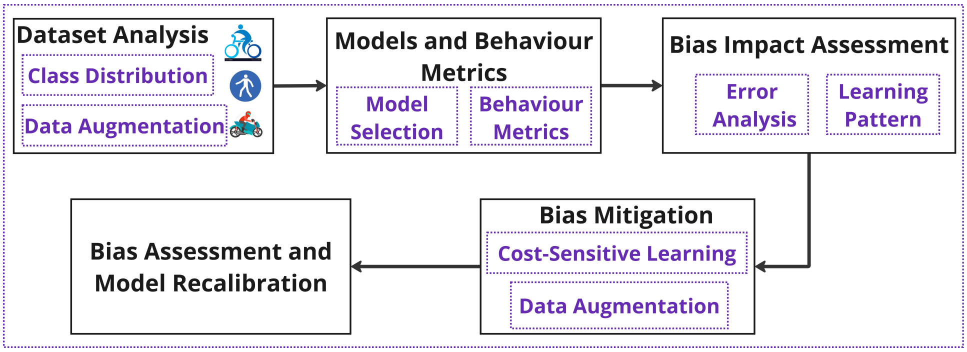


Figure 4.2: Proposed methodology for analyzing and mitigating biases

4.2.1. Dataset and Classes

The nuScenes dataset serves as the primary case study in this chapter, with emphasis on the Pedestrian, Cyclist, and Motorcyclist classes. However, the outlined approach can be applied and validated using other driving datasets, such as Waymo [113], where class definitions and distributions may vary.

A) Class Distribution Analysis: Statistical evaluation is conducted on the frequency of the targeted classes within the nuScenes and Waymo datasets to detect class imbalances. The datasets are further examined to assess representation in diverse contexts (e.g., urban and rural settings, varied weather and lighting conditions), thereby identifying underrepresented situations.

Pedestrian Class: The nuScenes dataset includes a substantial number of pedestrian annotations, with 149,921 labeled as "human.pedestrian.adult," constituting 21.61% of the total annotations. This extensive representation underscores the significance of pedestrian detection for autonomous driving. However, other pedestrian sub-categories, such as children (1,934 annotations, 0.28%), construction workers (13,582 annotations, 1.96%), and individuals using personal mobility devices (2,281 annotations, 0.33%), are less prevalent. These figures highlight limitations in dataset diversity among pedestrian subgroups.

Cyclist Class: Cyclists are annotated as "vehicle.bicycle" comprising approximately 17,060 instances (2.46% of the dataset). This lower occurrence relative to pedestrians influences class weight calculations and may affect the model's detection accuracy for cyclists, posing safety implications for urban environments.

Motorcyclist Class: The “vehicle.motorcycle" annotation accounts for around 16,779 instances, representing 2.42% of the dataset. While slightly less frequent than pedestrians, motorcyclists appear at a similar rate as cyclists. The limited representation of both cyclists and motorcyclists indicates an opportunity for enhancing class balance, especially in light of their distinct risks and operational dynamics.

Standard metrics utilized for evaluating model performance in nuScenes include *mean average Precision* (mAP), *Intersection over Union* (IoU), and the *nuScenes Detection Score* (NDS). mAP and IoU are widely used for 3D object detection and segmentation [40, 112], calculated on the basis of precision, recall, and the ratio of area overlap to intersection. The NDS metric integrates mAP with additional criteria to produce a holistic performance score for object detection models. The calculation for NDS is defined as:

$$NDS = \frac{1}{10}[5 \cdot mAP + \sum_{mTP \in TP} (1 - \min(1, mTP))] \tag{4.1}$$

Here mean Average Precision (mAP) measures the average precision across all object classes, combining precision and recall over multiple confidence thresholds to quantify detection accuracy and class balance. The Intersection over Union (IoU) defines the spatial overlap between predicted and ground-truth bounding boxes or segmentation masks, calculated as the ratio of the intersection area to the union area, and serves as a threshold criterion for identifying true positives during evaluation. The mean true positive error metrics (mTP) include normalized measures of translation, scale, orientation, velocity, and attribute errors, as defined in the nuScenes evaluation protocol [112]. Each component quantifies the deviation between predicted and ground-truth attributes, constrained within a range of 0 to 1. The nuScenes detection score (NDS) combines the mAP value with these mTP components to provide an indicator of overall detection performance, accounting for both classification accuracy and localization quality.

B) Data Augmentation and Re-sampling: Given the observed class distributions, especially the limited representation of certain pedestrian sub-categories, cyclists, and motorcyclists, dedicated strategies are applied to improve model training and evaluation.

Re-sampling Technique: To address class imbalances affecting groups such as cyclists and motorcyclists, resampling techniques are adopted to promote balanced training in autonomous driving models. Both random oversampling and undersampling are employed. In random oversampling, instances from minority classes are duplicated to increase their presence in the training data, allowing the model to learn from a more representative sample of these less frequent but significant road users. In contrast, undersampling reduces the number of instances from majority classes, for example, pedestrians, thereby limiting the influence of these groups during model training. These methods are implemented in a controlled manner to maintain data diversity [126].

Data Augmentation: Additional samples are created using geometric transformations, including image rotations and flips. Such augmentation enables a greater variety of class orientations to be represented in the dataset, which is essential for autonomous driving, where objects may be detected from multiple perspectives. This process helps prevent orientation bias and enlarges both the dataset’s size and diversity. Consequently, improvements in the robustness and generalization of models for perception tasks can be realized [127].

4.2.2. Models and Behavioural Metrics

Representative model architectures are used such as, ResNet18[128], SqueezeNet[129] from the CNN, Centerpoint[130], FS3D[131] from DNN and ViT [132] from the transformer family for evaluation. This selection enables a comparative evaluation of different architectures on biased datasets and supports an assessment of their generalization capacity following bias mitigation steps. These models offer insight into how design choices affect robustness across neural network frameworks.

Model Selection: ResNet18 [128] is widely used for perception applications and is effective in capturing spatial structures, although it may show bias toward common patterns or textures. SqueezeNet [129], characterized by its compact design, emphasizes extraction of global image features but may be sensitive to dataset diversity [115]. Vision Transformers (ViTs) analyze images by processing patches in sequence using attention mechanisms [132]. Although ViTs are effective at modeling dependencies, they may be less capable of identifying underrepresented classes or rare scenarios. The intrinsic learning dynamics of each model influence classification outcomes, underlining the necessity for specialized bias detection and mitigation protocols.

Behavioural Metrics: To detect biases in ResNet and SqueezeNet, neuron sensitivity and selectivity scores are measured. Sensitivity scores quantify the extent to which neuron responses vary with class-specific inputs, aiding the identification of potential biases. For example, a neuron in ResNet that responds more to vehicles than to cyclists could indicate vehicle-detection bias. Selectivity scores describe the degree to which a neuron's response is specialized for a given class, with higher selectivity indicating improved class distinction. This analysis allows for the identification of models' effectiveness in distinguishing classes and for detection of biases, such as increased selectivity for pedestrians over motorcyclists, implying bias in feature recognition [115]. The formula is defined as:

$$\text{Sensitivity Score} = \partial a / \partial x$$

where a is activation of neuron, and x is the input feature. The selectivity score is defined as:

$$\text{Selectivity Score} = \frac{a_c - a_{avg}}{\max(a_c, a_{avg})}$$

where a_c represents the activation of the neuron for the target class, and a_{avg} refers to the average activation across all classes.

For Vision Transformers (ViTs), the analysis of attention maps is important to understand which regions of the input image receive the model's focus and to identify potential sources of bias in decision-making. Attention weights are extracted from each transformer layer, reflecting how the model distributes its focus across image regions. These heatmaps, produced using the scaled dot-product attention mechanism, help detect imbalances in model focus that may indicate biased behaviour. The attention weights are computed as:

$$A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V,$$

where Q (Query), K (Key), and V (Value) are obtained by applying learned linear projections to the input embeddings of each image patch. Given an input embedding matrix

X, derived from the patch embeddings combined with positional encodings, the projections are computed as $Q = XW_Q$, $K = XW_K$, and $V = XW_V$, where W_Q, W_K, and W_V are trainable weight matrices. The term d_k is the dimensionality of the key vectors and is used for scaling to maintain stable gradients during training.

Analysing attention maps across layers and heads helps identify cases where the model places selective focus on specific features or neglects relevant regions, which can indicate biased patterns in the learned behaviour. Layer-wise Relevance Propagation (LRP) is another approach, beginning from the output layer and distributing the relevance of the predicted class backward through the network. In self-attention layers, LRP redistributes relevance scores to inputs based on the attention weights, allocating relevance among input patches [133]. Specific selectivity towards a patch in a layer, which shows the underlying basis of the model's prediction, is calculated as:

$$R_j^{(l)} = \sum_i A_{ij}^{(l)} R_i^{(l+1)} \tag{4.2}$$

Here $R_j^{(l)}$ is the relevance assigned to patch j in layer l, $A_{ij}^{(l)}$ denotes the attention weight from patch j to patch i within the self-attention structure, and $R_i^{(l+1)}$ represents the relevance in the subsequent layer.

4.2.3. Bias Impact Assessment

This section of the methodology (see Figure 4.2) is designed to uncover potential performance biases by examining model errors (including false positives and false negatives) as well as the propagation of bias related to class imbalances within learning patterns.

A) Error Analysis: Emphasis is placed on identifying false positives, such as instances where objects are wrongly classified into a target class, and false negatives, such as failure to recognize cyclists. *Class-specific error* analysis is conducted to assess model detection accuracy for pedestrians, cyclists, and motorcyclists under diverse scenarios. An evaluation is carried out to determine whether specific classes are consistently missed, and the subsequent influence on overall model accuracy is considered. Connections between these errors and previously measured bias indicators, such as neuron sensitivity and selectivity, are studied to reveal underlying causes and guide bias mitigation via *bias correlation*.

B) Learning Patterns and Bias Inheritance: Model architectures show unique learning tendencies and susceptibility to bias. For instance, convolutional neural networks (CNNs) such as ResNet may emphasize local textures and features, which can lead to bias if those features are unevenly distributed among classes, resulting in misclassification. By comparison, Vision Transformers (ViTs) may develop bias as a function of how attention is distributed across regions of the input image.

Bias Inheritance from Data: Model accuracy may be affected by the diversity and distribution of training samples. For example, models exposed predominantly to pedestrian data collected in urban environments may perform inadequately when applied to rural settings. Addressing this type of bias inheritance is necessary for building perception systems that generalize across various real-world environments, making it important to refine both sampling and training processes.

C) Comparative Analysis: ResNet, SqueezeNet, and ViTs are evaluated on identical datasets to detect specific biases inherent to each approach. This comparative process facilitates the categorization of models according to their bias tendencies, supporting informed selection for scenarios where minimizing bias is essential.

Interpretation and Visualization: Layer-wise Relevance Propagation (LRP) is employed to identify which features of the input influence the models output. If a model is observed to frequently rely on non-essential background features, this is regarded as an indicator of bias that requires correction [134]. Visualization methods such as heatmaps and attention maps are applied to interpret model behavior and assist in refining the training process, with the aim of reducing bias. Collectively, this stage comprises the analysis of learning patterns, the study of bias inheritance from data, comparative evaluation of different architectures, and visualization techniques to assist in recognizing and addressing sources of bias.

4.2.4. Bias Mitigation Technique

Several techniques are investigated to mitigate bias introduced by imbalanced class representation. These strategies are designed to promote equitable and consistent model outcomes.

A) Cost-Sensitive Learning: The model loss function is adapted using cost-sensitive learning, which incorporates class representation to address imbalance in the training process. Details of class weight calculation and the adjustment to the loss function are described below.

Class Weights Calculation: Weights for the pedestrian (w_p), cyclist (w_c), and motorcyclist (w_m) classes are obtained by computing the inverse of their respective proportions in the dataset. This method increases the influence of minority classes during training. The weights are calculated as:

$$w_p = \frac{1}{21.61\%}; \qquad w_c = \frac{1}{2.46\%}; \qquad w_m = \frac{1}{2.422\%} \tag{4.3}$$

These weights are then normalized to ensure they contribute proportionally and maintain stability during training.

Loss Function: These precalculated weights are then integrated into the model's loss function, utilizing a weighted multi-class comprehensive cross-entropy loss. For each instance *i*, the loss is defined as:

$$L_i = -w_p y_{ip} \log(p_{ip}) - w_c y_{ic} \log(p_{ic}) - w_m y_{im} \log(p_{im})$$

Here w_p, w_c, and w_m are the weights for pedestrians, cyclists, and motorcyclists. $y_{i(x)}$ is a binary indicator of whether instance i belongs to class x. $p_{i(x)}$ is the model's predicted probability for instance i belongs to a class x.

Implementing focal loss [135] provides another practical mechanism that focuses on hard-to-classify instances by reducing the loss contribution from easily classified examples. This dynamic adjustment is controlled through a tunable focusing parameter, which leads to faster and more effective learning on datasets where intra-class variation in example difficulty is significant. By mitigating the influence of numerous easy examples, Focal Loss can enhance the model's learning efficiency, allowing for a more nuanced understanding and handling of complex class imbalances. However, the weighted loss functions, particularly weighted cross-entropy, offer a straightforward and easily implementable solution. By assigning weights inversely proportional to class frequencies, they directly compensate for imbalances, ensuring that minority classes have a proportionally higher influence during the training process. This method simplifies the implementation for the models selected for experimental evaluation and further enhances the stability and predictability of the training process, making it a robust choice for preliminary model training where simplicity and direct addressing of class imbalances are prioritized. Therefore, the selection between these two loss functions depends on the dataset's requirements and the desired balance between disparities and dynamic learning efficiency.

Dynamic Adjustment Evaluation: Continual adaptability of the model is supported by dynamically modifying class weights in response to performance metrics. This adjustment can be evaluated on a validation set, monitoring whether increased accuracy for underrepresented classes is achieved without leading to overfitting.

Overall, applying this cost-sensitive learning strategy helps reduce biases caused by class imbalance in the dataset, leading to more balanced and consistent performance across all categories. This approach strengthens the reliability of the model and improves trust in perception tasks, particularly in the safety-critical settings.

B) Data Augmentation Guided by Model Analysis: A data augmentation strategy is adopted to correct model biases, particularly those detected in Vision Transformers (ViTs) through the examination of attention maps and Layer-wise Relevance Propagation (LRP). This approach specifically addresses the limited presence of cyclists and motorcyclists within the nuScenes dataset.

Attention-Guided Augmentation: Patterns observed in attention maps guide the augmentation procedure. For example, if the model repeatedly fails to identify pedestrians under certain lighting, this highlights a need for additional training samples representing those conditions. Techniques such as zooming, altering brightness, or adjusting contrast are used to diversify these challenging scenarios. Likewise, new examples featuring diverse poses or orientations for motorcyclists are included to improve detection.

LRP-Informed Sampling: Layer-wise relevance propagation (LRP) provides insight into which visual features most strongly influence the model's predictions. LRP works by backpropagating the model's output score through the network and distributing this score across the input pixels according to their contribution to the predicted result. In doing so, it produces heatmaps that highlight the regions of the input that the model relies on most. Based on these explanations, additional samples are added to the training set to strengthen the representation of features that are important for correct identification. This augmentation is applied to instances where the model misclassifies images, such as those containing occluded subjects or distinctive textures, thereby improving the model's ability to distinguish between informative and misleading features.

Implementation: This approach is implemented for dataset refinement process as a continuous process, informed by model analysis. Aligning data augmentation directly with the observed model behaviour helps to reduce biases and promote more balanced model outcomes. This methodology is essential for perception systems, where accurate detection of all categories of road usersincluding those who are infrequently observed, such as cyclists and motorcyclistsis necessary.

4.2.5. Bias Assessment and Model Re-calibration

Mitigating bias in AI models relies on continual evaluation and recalibration. The process described below outlines the framework for bias assessment and model adjustment.

Bias Assessment: Behavioural metrics, such as sensitivity and selectivity scores for each class, are employed to gauge the impact of mitigation efforts. By comparing these metrics before and after interventions, changes in the models learning patterns and focus can be identified. Error rates, especially false positives and false negatives, are also analyzed to evaluate improvements in class-specific predictive accuracy. In the case of Vision Transformers, attention maps are reviewed to assess whether attention is now more equitably distributed across all classes and relevant scenarios.

Model Re-calibration: Class weights in the loss function are dynamically tuned using real-time performance metrics. This step ensures that the model remains responsive to any variations in class representation or shifts in dataset composition. When additional class samples or new data are introduced, retraining is performed to maintain model relevance. Iterative refinement follows, where the impact of the latest bias mitigation actions is continually monitored and strategies are updated as needed.

4.3. Evaluation on Vulnerable Classes

This section details the training, testing, and evaluation methodology. The full dataset consists of around 1.4 million images, annotated across several categories. The Pedestrian (149,921), Cyclist (17,060), and Motorcyclist (16,779) classes serve as the focus, offering valuable insight into object detection bias.

Preprocessing Steps: Data preprocessing includes normalization of pixel values to the [0,1] interval, consistent alignment of annotation formats, and resizing of all images to standardized input dimensions suitable for each model.

Model Architectures: The study involves SqueezeNet and ResNet18 from the CNN family, alongside Vision Transformer (ViT). SqueezeNet is noted for its compact structure and robust feature extraction, while ResNet18 is selected for its residual learning capacity. ViT is incorporated to investigate the impact of attention-based mechanisms on addressing class imbalance.

1) Hyperparameter Optimization: Initial hyperparameter settings are as follows: learning rate of 0.001, batch size of 32, and weight decay of 0.0001 for CNN models. For ViT, training is configured with a batch size of 32, starting learning rate of 1e-3 linearly decaying to 1e-5, and a 30-epoch training period using the Adam optimizer. Dropout is set at 0.1, and weight decay at 0.03, to mitigate overfitting.

Optimization Techniques: A grid search approach is adopted for hyperparameter tuning, with systematic adjustment of learning rates, batch sizes, and dropout rates to determine optimal combinations [123]. The best set is chosen according to the highest mean detection scores across metrics, following the evaluation protocols in [112].

2) Training Process: Due to the relatively small class sample size, CNN models are trained for 50 epochs with the Adam optimizer. The learning rate is reduced by 10% every 10 epochs using a scheduler. Regularization techniques, including dropout and data augmentation (rotations and flipping), are employed to control overfitting [136].

The dataset is split into 70% for training, 15% for validation, and 15% for testing, providing an equitable distribution of scenario types in each subset. Training utilizes a high-performance computing cluster equipped with NVIDIA Tesla V100 GPUs, with Open MPI and PyTorch as the main frameworks for deep learning.

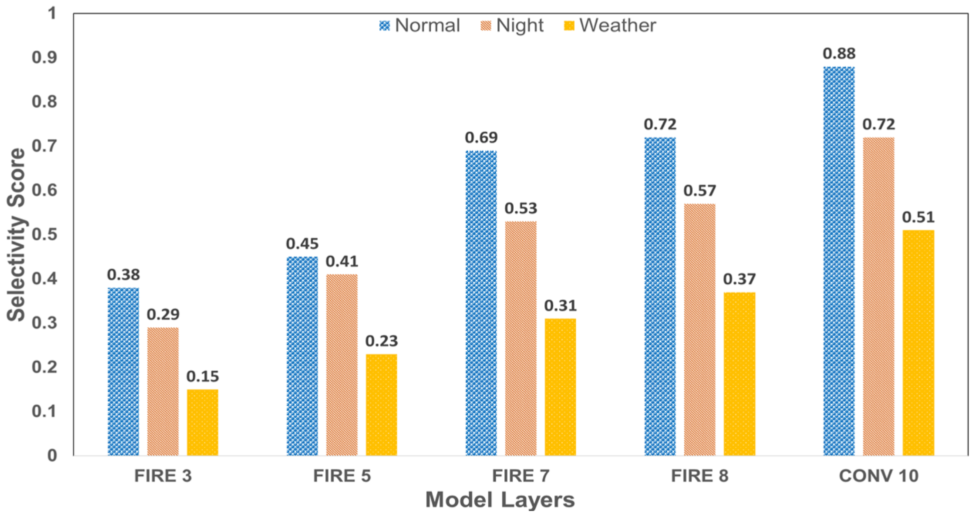


Figure 4.3: Layerwise selectivity for the Classes on SqueezeNet Model

3) Model Evaluation: Model evaluation employed mAP, percentage IoU, and NuScenes Detection Score (NDS), as well as sensitivity and selectivity scores to analyze class behaviour.

Baseline Comparison: Performance was compared before and after applying bias mitigation strategies. Baseline models were trained using the original dataset conditions, providing a point of reference for subsequent improvements.

Class-specific Analysis: Performance metrics were examined for each class to assess the impact on detecting pedestrians, cyclists, and motorcyclists. Attention was focused on shifts in false positive and false negative rates for these groups. For CNNs, analysis included layer-wise investigation and data diversity testing, especially when one class was more prevalent within the dataset.

Layer-wise Analysis: This analysis determines if certain layers show a preference toward particular classes, which informs strategies for bias mitigation. For example, when initial layers display stronger activation for pedestrians compared to cyclists or motorcyclists, it becomes necessary to adjust the training dataset for better inclusiveness. Figure 4.3 presents selectivity scores across SqueezeNet layers under various conditions: Normal, Night, and Weather. The results show that deeper layers have higher selectivity, with the highest scores in standard conditions, and lower scores observed during

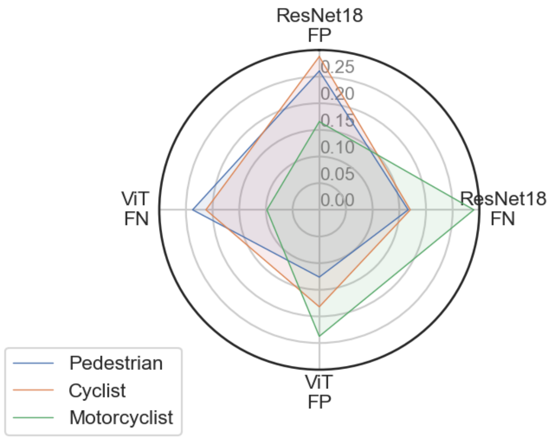


Figure 4.4: Class-wise error rates (false negatives and false positives) for ResNet and ViT during model validation.

Night and Weather scenarios. A comparable approach is used at the epoch level, where performance metrics, training loss, and accuracy are examined alongside behavioural metrics, offering insights into the evolution of feature learning.

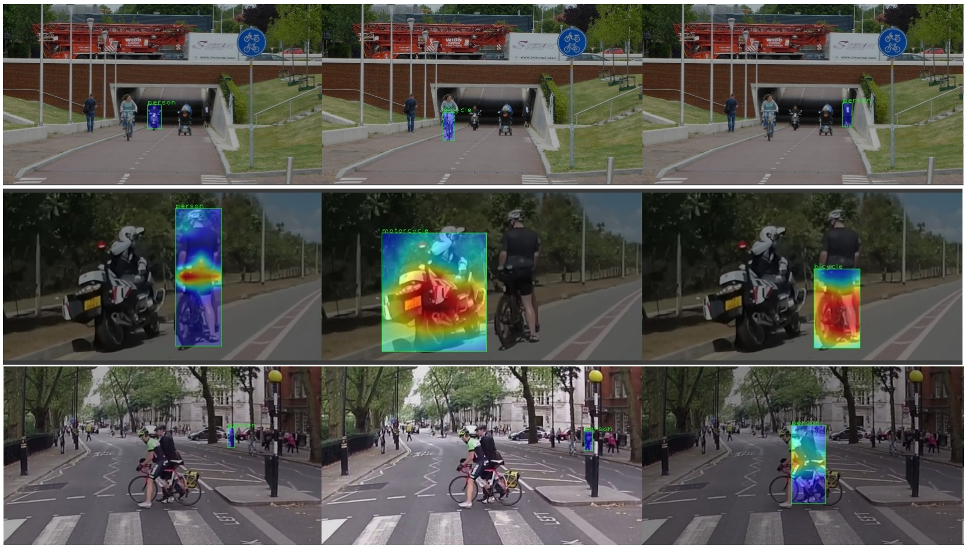


Figure 4.5: Figure showing heatmap for all three classes.

4) Bias Impact Analysis: To assess models as outlined in the methodology, the following section covers error analysis and learning patterns, using heat maps and layer-wise visualizations.

Error Analysis: Figure 4.4 provides error analysis for the classes under study. ResNet18 displays a greater tendency for false positives across all classes compared to ViT, which points to a likelihood for overprediction. ViT gives fewer false positives but a higher number of false negatives, especially for Pedestrians and Cyclists, suggesting limitations in detection for these categories. For the Motorcyclist class, ViT achieves more balanced detection with reduced false negatives relative to ResNet18, indicating improved reliability.

Visualizations: Figure 4.5 includes heatmap visualizations, highlighting the focused regions of models selectiveness towards a class, its feature, and differences before mitigation strategies. Training on unbalanced classes results in both CNN models misclassifying motorcyclist and cyclist categories.

5) Vulnerable Road User Analysis with Vehicle Class: Since the Vehicle (Car) class constitutes approximately 36.6% of the nuScenes dataset, a detailed evaluation is performed by including this class along with vulnerable road users. Accordingly, class weight calculations are modified as follows:

$$w_v = \frac{1}{36.6\%};\; w_p = \frac{1}{21.6\%};\; w_c = \frac{1}{2.4\%};\; w_m = \frac{1}{2.4\%}$$

Table 4.1: NDS and %IoU of VRU with ResNet18.

%IoU of Classes			NDS	Data Condition
Pedestrian	Cyclist	Motorcyclist		
86.1	53.6	76.4	72.0	Normal
78.5	8.3	40.7	42.5	Night
67.3	7.5	18.9	31.2	Weather
85.7	52.8	76.1	71.5	Rotated
72.1	12.4	21.4	35.3	Mixed

4.3.1. Analysis of Results

For evaluating IoU, mAP, and NDS scores reported in Table 4.1 and Table 4.3, a subset approach was applied. In this setup, one class (such as pedestrian) maintained a representation of 67%, while the other two classes were equally distributed. This proportion was iteratively adjusted so that each class achieved equal representation over the course of testing. The intent was to observe metric behaviour under conditions where class weights are normalized. Table 4.1 presents Intersection over Union (%IoU) for each class and NuScenes Detection Score (NDS) under various conditions for ResNet18. Pedestrian detection remains strong, with IoU values above 70% in all conditions except during combined adverse scenarios. Cyclist detection, in contrast, drops considerably outside of normal conditions, reaching as low as 7.5% IoU during poor weather. Detection for motorcyclists is also influenced by adverse scenarios, though the reduction is less pronounced than for cyclists. The general trend is reflected in the overall NDS, which peaks at 72.0 under normal conditions and falls to 31.2 when weather is unfavourable. Rotated conditions have minimal effect on IoU for pedestrians and motorcyclists, but a more noticeable impact on cyclists. In mixed conditions, all classes experience a decrease in IoU, with the corresponding NDS dropping to 35.3.

Table 4.2: NDS and %IoU for classes with ResNet18.

%IoU of Classes				NDS	Data Condition
Car	Pedestrian	Cyclist	Motorcyclist		
77.0	68.1	53.6	56.4	64.9	Normal
73.1	64.5	8.9	38.7	45.6	Night
72.8	61.3	7.8	20.9	39.2	Weather
76.7	67.9	52.8	56.1	62.5	Rotated
71.4	65.1	12.9	17.6	36.8	Mixed

Table 4.3: mAP of VRUs With SqueezeNet.

Pedestrian	Cyclist	Motorcyclist	Total	Data Condition
86.3	54.7	77.8	72.9	Normal
78.1	7.5	38.3	41.3	Night
67.0	8.3	19.7	31.6	Weather
85.9	52.3	75.2	71.1	Rotated
73.2	14.8	22.1	36.7	Mixed

Table 4.3 provides the mean average precision (mAP) scores for detecting pedestrians, cyclists, and motorcyclists across various conditions. Under standard test scenarios, pedestrian detection is high at 86.3%, with cyclist detection at 54.7%. Motorcyclist detection reaches 77.8%, resulting in an overall mAP of 72.93%. Performance declines during night and adverse weather, with cyclist detection falling as low as 7.5% and 8.3% respectively. When normal condition images are rotated, detection of pedestrians and motorcyclists drops marginally (about 1%), but cyclist performance declines to 52.3%. In the presence of combined challenging conditions, the overall mAP reduces to 36.7%, highlighting the challenges these models encounter in more complex scenarios.

Table 4.4: Class-specific mAP performance of SqueezeNet in various conditions.

Car	Pedestrian	Cyclist	Motorcyclist	Total	Condition
76.1	67.3	8.7	27.8	44.9	Normal
74.7	58.1	5.5	22.3	39.9	Night
73.2	49.0	4.3	19.7	35.7	Weather
75.8	65.6	7.9	21.2	42.1	Rotated
74.9	61.8	4.6	24.1	40.8	Mixed

As shown in Table 4.2 and Table 4.4, including the vehicle class in evaluations using ResNet18 and SqueezeNet changes the performance trade-off under different data conditions. The prevalence of vehicles influences how well vulnerable road users (VRU) such as cyclists and motorcyclists are detected, especially in more difficult settings like night or inclement weather. While normal and rotated scenarios retain high performance across classes, a decrease in %IoU and NDS is seen when vehicles are included, indicating increased model selectivity for the most represented class. Notably, cyclist and motorcyclist performance declines more in night, weather, and mixed scenarios, pointing to a tendency for the model to favour the vehicle class. This outcome suggests a need to adjust class weighting or apply more sophisticated augmentation methods to maintain balanced detection, especially for less represented classes.

Table 4.5: Class-wise Performance on the CNN models.

Class	Model	%IoU	NDS	Sens.	Sel.
Pedestrian	ResNet18	65.3	64.1%	0.74	0.78
Cyclist	ResNet18	68.4	79.5%	0.65	0.67
Motorcyclist	ResNet18	70.2	80.3%	0.70	0.72
Car	ResNet18	75.1	78.1%	0.78	0.83
Pedestrian	SqueezeNet	78.1	85.4%	0.81	0.83
Cyclist	SqueezeNet	72.6	82.7%	0.73	0.75
Motorcyclist	SqueezeNet	74.0	83.2%	0.76	0.77
Car	SqueezeNet	78.1	85.4%	0.81	0.83

Sens. = Sensitivity Score, Sel = Selectivity Score

Class-wise Performance Metrics: Table 4.5 lists the class-specific scores for ResNet18 and SqueezeNet. SqueezeNet achieves higher %IoU and NDS for pedestrians, cyclists, and motorcyclists, and show higher sensitivity and selectivity in distinguishing among these classes, suggesting better recognition and fairness in its predictions.

Baseline vs Post-Mitigation Performance: Table 4.6 compares Intersection over Union (IoU) and NuScenes Detection Score (NDS) for ResNet18 and ViT models before and after bias mitigation. Both models improve in %IoU and NDS after mitigation, with ResNet18 showing a 4.3% gain in IoU and a 3.1% gain in NDS, and ViT improving by 4.3% and 3.3% respectively. These results confirm the effectiveness of bias mitigation techniques in boosting detection outcomes.

Table 4.6: Baseline vs Post-Mitigation Performance.

Model	Metric (Avg)	Baseline	Post Mitigation
ResNet18	%IoU	71.3%	75.6%
ViT	%IoU	74.9%	79.2%
ResNet18	NDS	80.6%	83.7%
ViT	NDS	83.8%	87.1%

To further evaluate the methodology beyond nuScenes and Waymo, we also perform class-wise performance on the Argoverse2 dataset [137], which contains diverse urban environments and varying distributions of pedestrian, cyclist, and motorcyclist samples. Argoverse2 serves as an additional benchmark for studying class imbalance and assessing how mitigation techniques influence multi-class detection performance. Table 4.7 compares the performance of CenterPoint and FS3D models on the Argoverse2 dataset, pre- and post-mitigation. CenterPoint's mAP increases from 37.4% to 45.2%, and gains are also seen in individual class detection rates. FS3D's mAP rises from 49.4% to 51.9%, with improvements especially marked for motorcyclists. This trend further supports the advantage of mitigation approaches in refining detection accuracy and fairness.

Table 4.7: Performance analysis on Argoverse2.

Models baseline performance				
Method	**APped**	**APcyc**	**APmotor-cyc**	**mAP**
CenterPoint	48.6	26.5	37.1	37.4
FS3D	61.4	34.5	51.7	49.2
Post mitigation performance				
CenterPoint-1	54.8	38.8	42.0	45.2
FS3D-1	62.8	36.4	56.7	51.9

Table 4.8 presents average precision metrics for pedestrian, cyclist, and motorcyclist classes using CenterPoint and FS3D across different class balance scenarios. With modified training samples, improvements in sensitivity and selectivity especially for pedestrians and motorcyclists are recorded. Cyclist detection, however, shows only limited gains, highlighting persistent difficulties in this class.

Table 4.8: Class Performance on the Argoverse2.

Class	Model	AP	Sens.	Sel.
Pedestrian	CenterPoint	48.6	0.73	0.77
Cyclist	CenterPoint	26.5	0.63	0.68
Motorcyclist	CenterPoint	37.1	0.69	0.74
Pedestrian	FS3D	61.4	0.77	0.82
Cyclist	FS3D	34.5	0.64	0.69
Motorcyclist	FS3D	52.5	0.70	0.78
Pedestrian	CenterPoint-1	54.8	0.81	0.82
Cyclist	CenterPoint-1	38.8	0.62	0.69
Motorcyclist	CenterPoint-1	42.0	0.73	0.75
Pedestrian	FS3D-1	62.8	0.85	0.89
Cyclist	FS3D-1	36.4	0.65	0.70
Motorcyclist	FS3D-1	56.7	0.76	0.80

Sens. = Sensitivity Score, Sel = Selectivity Score

Table 4.9 includes pre- and post-mitigation results for the Waymo dataset, comparing vehicle, pedestrian, and cyclist detection. Post-mitigation, both CenterPoint-1 and FS3D-1 shows metric improvements, such as higher APH for vehicles and better scores for pedestrians. Cyclist performance, though improved, remains less responsive to mitigation, consistent with other datasets.

Table 4.10 shows results from the ML performance metrics Precision, Recall, and Mean Average Precision (mAP) for classes including Cars, Pedestrians, Cyclists, and Motorcyclists as evaluated using the CenterPoint model. Before mitigation efforts, the Car class has the highest Precision (0.88) and mAP (0.80), showing the model's ability to accurately identify and predict car class and subclass. Pedestrians class which has second

Table 4.9: Performance analysis on Waymo Level 1.

AP/APH	Vehicle	Pedestrian	Cyclist	mAP/mAPH
CenterPoint	75.1/ 77.6	78.2/ 74.9	71.8/ 70.4	75.0/ 74.3
FS3D	77.5/ 77.2	80.9/ 74.2	76.1/ 75.3	78.1/ 75.5
Post mitigation performance				
CenterPoint-1	77.6/ 78.1	80.5/ 76.2	73.1/ 70.6	77.0/ 74.9
FS3D-1	78.1/ 77.5	81.6/ 75.2	77.0/ 76.1	78.9/ 76.2

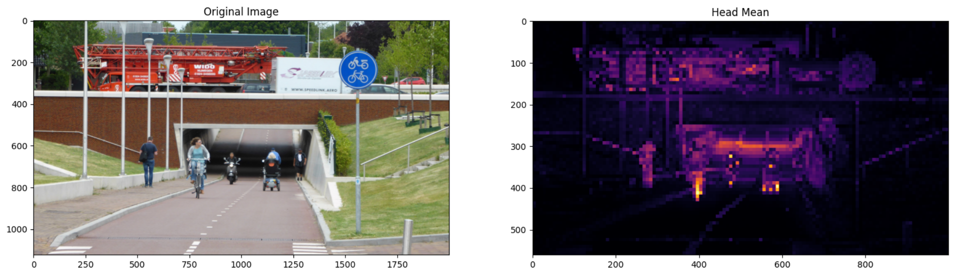


Figure 4.6: Figure showing mean for all three classes

highest presence in the data subset also showed robust model performance with Precision at 0.83 and mAP at 0.69. However, Cyclists and Motorcyclists show significantly lower metrics, with Cyclists having low mAP of 0.47, showing the challenges the model faces in handling less represented or out-of-domain classes. After implementing mitigation strategies, all classes has a decrease in Precision and Recall, suggesting that the mitigation process affects the model's sensitivity and specificity. For e.g., the Precision for Cars decreased to 0.85 and mAP to 0.77, while Pedestrians has a reduction in Precision to 0.81 and mAP to 0.63. Cyclists and Motorcyclists shows lower performance metrics post-mitigation, with Cyclists' mAP further reducing to 0.45. These changes show the model's challenge in accurately detecting less represented classes even post-mitigation, highlighting the need for more refined strategies that can enhance performance without compromising detection accuracy across all classes.

Visualizations: Figure 4.7 and Figure 4.8 display mean and attention maps focused on the motorcyclist class, illustrating that features related to cyclists may still be emphasized in the presence of cyclist instances within the input. In these tests, the model shows dominant weighting toward the motorcyclist category, which results in an inability to effectively recognize cyclist features. Similarly, Figure 4.6 and Figure 4.9 present test results where all classes are equally represented in the input; however, the attention maps reveal that the model tends to associate features from cyclists and pedestrians with other class predictions. This behaviour indicates incomplete class separation, even in balanced subsets.

In summary, the following observations are made:

- In CNN models, error analysis identifies a higher incidence of false positives in ResNet18 compared to SqueezeNet.

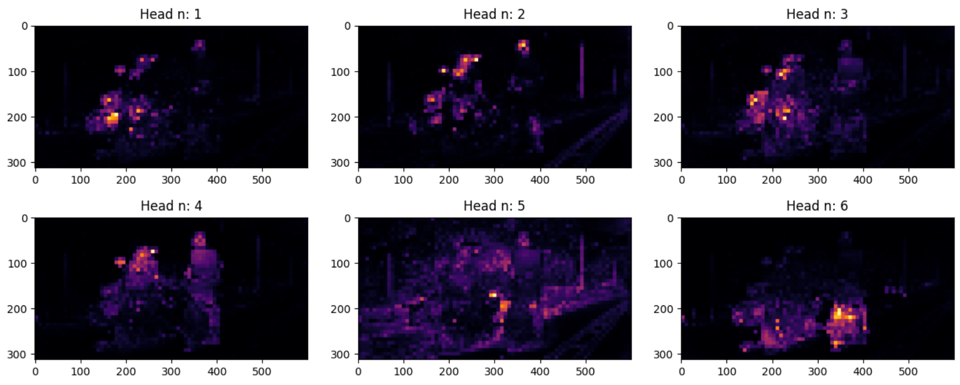


Figure 4.7: Head-wise attention maps for motorcyclist class.

Table 4.10: ML Performance Metrics Comparison.

Class	Model	Precision	Recall	mAP
Car	CenterPoint	0.88	0.76	0.80
Pedestrian	CenterPoint	0.83	0.75	0.69
Cyclist	CenterPoint	0.58	0.42	0.47
Motorcyclist	CenterPoint	0.55	0.39	0.41
Post mitigation performance				
Car	CenterPoint	0.85	0.70	0.77
Pedestrian	CenterPoint	0.81	0.68	0.63
Cyclist	CenterPoint	0.57	0.34	0.45
Motorcyclist	CenterPoint	0.52	0.31	0.37

mAP = Mean Average Precision

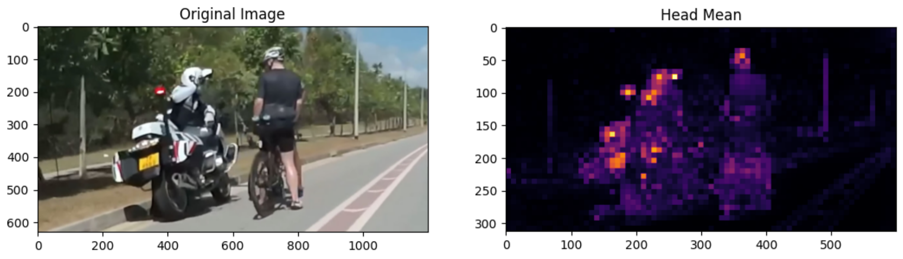


Figure 4.8: Figure showing mean for motorcyclist classes.

- After applying mitigation strategies such as resampling, both ResNet18 and ViT shows improved detection outcomes.
- Across different testing scenarios, pedestrian detection remains most reliable among vulnerable road user classes; however, the overall post-mitigation performance shows a slight reduction, suggesting a need for further improvements in model

robustness.

- Even when class samples are balanced within a subset, the behaviour metrics during model training may still reflect unequal representation across classes.

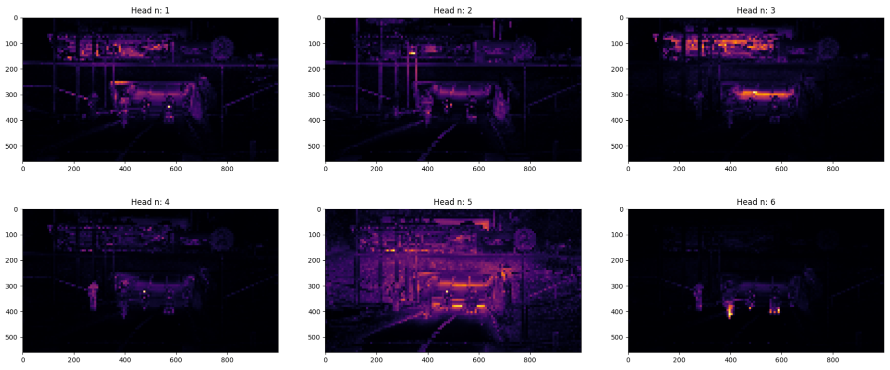


Figure 4.9: Head-wise attention weights for all classes.

4.3.2. Methodology Adaptation to Other Datasets:

To adapt the proposed methodology to other visual datasets (such as Lyft, A2D2, etc.), the first step is to identify class imbalances or overrepresentation in the input data using class distribution analysis, which involves statistically evaluating class frequency to ensure fair representation, as biases might arise due to more frequent appearance of particular objects in specific environments. The next step will be to detect these biases using behavioural metrics on the trained model: sensitivity and selectivity. For example, in a dataset like CIFAR-10, where all ten classes have equal representation, the behavioural metrics help to identify potential biases arising from the class's local features, texture, and colour, while in other datasets such as Lyft or A2D2, the focus will be on the models' ability to detect different object sizes and their spatial relationships accurately. The third step is implementing mitigation strategies, including resampling techniques, using oversampling to increase the presence of underrepresented classes and undersampling to decrease over-representation. While CIFAR-10 may need simple adjustments due to its uniform image sizes, Lyft and A2D2 datasets could benefit from more complex data augmentation using rotation, flipping, and adding image corruptions to existing samples of CIFAR-10 and using generative adversarial networks to represent and add complex scenarios to Lyft and A2D2. The last step in adapting the methodology is implementing cost-sensitive learning to modify the training loss functions using weighted calculation (of class in the dataset) and adding it to the proposed loss function.

4.4. Conclusion

Random model compression and approximation techniques often prune or approximate model weights based on learned feature weights, which can lead to the loss of critical information in less-represented or more complex features, potentially embedding biases or vulnerabilities. However, we can strategically apply approximation strategies by integrating behavioural metrics analysis into the dataset preparation and model training processes. This ensures that essential and sensitive features are preserved, leading to more robust and fair AI models that maintain high performance while reducing computational demands. As compared to operational interventions, which often target immediate, model-specific optimizations or parameter adjustments, a strategic approach considers the broader, long-term objectives and underlying patterns that influence model behaviour across varied datasets and applications. By introducing behavioural metrics analysis and bias assessment at the design stage, this work aligns model approximation and bias mitigation with high-level goals such as fairness, adaptability, and sustainable deployment. The advantage of the used approach is reflected in the experimental results across nuScenes, Waymo, and Argoverse2, where the mitigation strategies improve class-wise detection performance even under different class distributions, scene conditions, and sensing scenarios. These consistent gains show that early-stage behavioural analysis supports solutions that remain reliable as the data characteristics and deployment environments evolve.

This chapter examined the challenge posed by class imbalance in driving datasets, with a particular focus on underrepresented classes and their influence on AI model performance. The discussion addresses research question 2 and the artefact pertaining to bias evaluation and mitigation. The experiments conducted with widely used CNNs, DNNs, and Vision Transformers shows that biases present in datasets can result in divergent learning behaviours, thereby impacting the accuracy and reliability of perception systems. In addition, various bias mitigation strategies, including cost-sensitive learning and targeted data augmentation, were implemented and assessed. These methods contributed to improved detection results, particularly for Intersection over Union (IoU) and NuScenes Detection Score (NDS) metrics. When the proposed mitigation approaches were applied, IoU(%) and NDS(%) for CNN models increased from 71.3 to 75.6 and from 80.6 to 83.7, respectively. For Vision Transformers, improvements were observed in IoU and NDS metrics from 74.9 to 79.2 and from 83.8 to 87.1, showing the effectiveness of these bias reduction measures. The outcome of this chapter, based on the behavioural metrics, bias impact assessment, and class disparities studies, forms the fundamental block addressing model performance and robustness. The usage of behavioural metrics improves knowledge of model learning process and understanding of feature extraction and decision-making process on the granular level, which further provides analysis for implementing approximate strategies, for example, layer-wise bit-width reduction, reduced multipliers on computational complex layers of ML models, which is part of model approximation strategies and comprehensively covered in next chapter.

5

AxCEdge: Approximate Computing for the Edge

Energy-aware computing provides mechanisms to lower computational energy use by applying both hardware and software optimization techniques. One such hardware approach is to lower the processor's frequency. However, this will reduce the on-board clock speed, and the same process will take extra time for computation, thus bringing a potential **trade-off**. As the research scope is on energy-aware computation within the application domain, the exploration will be on software and model-based **approximate computing** to develop approximation schemes for data-intensive vehicle services as an element or component of the proposed energy-aware adaptive framework. Approximate computing practices include relaxing the need for full precise operation using parameter approximation (function). The parameters are traded or sacrificed for applications and services resilient to accuracy and data loss, which means 'quality acceptable to perform the application accurately, despite approximation'. In connected vehicular services, several perception and non-safety-critical applications fall into this category, making them suitable for energy-aware approximation methods.

Within this context, approximation mechanisms can reduce computational demand and help lower on-board energy consumption. This leads to the research question (RQ2): Which components can enable task deployments energy-efficiently and collaboratively in vehicle-edge-cloud computing?. Building upon the knowledge of model learning and representation from the previous chapter, this chapter contributes to RQ2 by proposing and evaluating a few approximation schemes on perception tasks. By proposing these mechanisms for CNN, DNNs and ViTs, this chapter aims to develop approximation components of the framework. The chapter also covers the implementation of approximation schemes combined with mixed-precision techniques in quantization-aware training and post-training quantization from the software approximation perspective.

By strategically applying software approximation strategies, such as approximate multipliers, reduced-precision operations, and variational inference, to deep learning models (CNNs, DNNs, and Vision Transformers) used in vehicular services, it is possible to reduce computational complexity and energy consumption without significantly compromising model performance for real-world applications. [1] Different approximation techniques may be more suitable for specific model architectures. For example, approx-

[1] Published as: Katare, et. al. "Energy-efficient edge approximation for connected vehicular services." 2023 57th Annual Conference on Information Sciences and Systems (CISS). IEEE, 2023.

imate multipliers might be effective for CNNs and DNNs, while variational inference could be well-suited for Vision Transformers. In this chapter, multiple software-level approximation strategies are explored as schemes (e.g., scheme 1 is a probabilistic approximation of convolutional layers, and scheme 2 is a linear approximation on models using multipliers) and each of these schemes is tested and evaluated on vision models, and datasets to develop multiple approximation strategies for models which can be integrated within the framework. The contributions of this chapter are as follows:

- We propose four approximation schemes using probabilistic approximation, approximate multipliers, and variational inference combined with mixed-precision quantizations and evaluate their impact on model performance parameters. [2]
- Using the DNNs and Transformers, we further explore and analyze the trade-off for training and inference of these approximation schemes on edge devices. [3]

As CNNs, DNNs and ViTs architectures are supported by common and unique modules and operations such as convolution, fire modules, fully connected, and multi-head, self-attention mechanisms, we first discuss approximation schemes suited for these modules and models by providing a high-level operational overview of proposed methods. Each of these proposed approximation schemes includes empirical tests and analyses. The following sections discuss how adaptive approximation techniques improve and optimize models for efficient computing in a resource-constrained environment.

5.1. Scheme 1: Approximate Convolutions

With the advancements in separable convolution, depth-wise convolution, and residual blocks, the deployment of deep neural network models on resource-constrained embedded devices has been widely adopted [138]. Such methods have made it possible for complex tasks to be executed in optimized conditions on these platforms. An efficient model selected for semantic mapping in autonomous vehicles is BiSeNet [139], which has been designed as a lightweight architecture for real-time applications. The structure of the BiSeNet layers is depicted in Figure 5.1. BiSeNet [139], first introduced in 2018, was designed specifically for real-time image segmentation. This model consists of two main components: the Spatial block and the Context block, as summarized in Figure 5.1. The Spatial block contains convolution, batch normalization, and ReLU layers. Its purpose is to retain high-quality features and spatial information from the input, which are preserved for later feature-based fusion with the output of the Context block. For semantic segmentation, the receptive field is essential for incorporating relevant features into convolutional networks. In the BiSeNet architecture, the Context block is responsible for supplying the receptive field. This block consists of average pooling, convolution, ReLU, Sigmoid operations, and a global average pooling layer, resulting in a maximized receptive field for features and a lightweight overall structure. The computational cost associated with these tasks is addressed by a refinement module, which serves to enhance

[2]Published as: Katare, et al. "Approximating Vision Transformers for Edge: Variational Inference and Mixed-Precision for Multi-modal Data." Springer Nature Computing (2025)

[3]Katare, et. al. "Approximation Strategies for Vision Models on Edge Devices: An Accuracy-Efficiency Trade-off". IEEE TPAMI (Under Review as of January 2025)

the features extracted from the input image at each stage. This refinement module is implemented through average pooling and the use of vector maps that store the learned features. Such a module can be readily integrated for feature-based fusion, without the need for up-sampling, which further reduces computational requirements. A high-level overview of the architecture is presented in Figure 5.1.

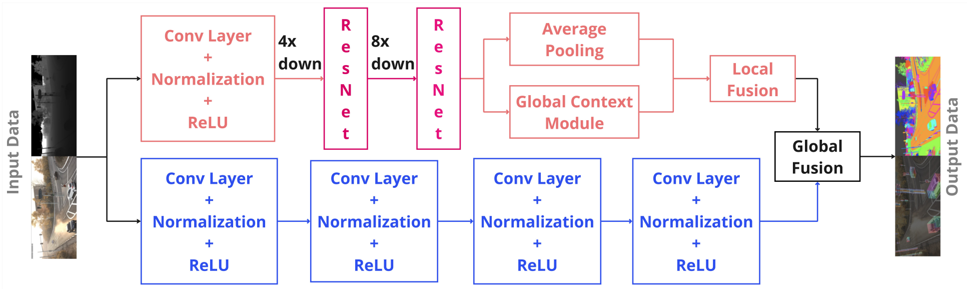

Input Data
Conv Layer + Normalization + ReLU
4x down
ResNet
8x down
ResNet
Average Pooling
Global Context Module
Local Fusion
Global Fusion
Conv Layer + Normalization + ReLU
Conv Layer + Normalization + ReLU
Conv Layer + Normalization + ReLU
Conv Layer + Normalization + ReLU
Output Data

Figure 5.1: DNN architecture with multiple convolutional layers for semantic map.

Convolutional layers are selected due to their compute-intensive characteristics [140]. As shown in the architectural layout (Figure 5.1), the approximation addressed in this section includes both probabilistic approximation and the reduction of multiplications in the convolutional layer. For a given system, the relative error between the original model $f[x]$ and the approximated model $g[x]$ can be determined as:

$$E(g,f) = \sum_{n} \left| \frac{g[x] - f[x]}{f[x]} \right| \tag{5.1}$$

As the architecture incorporates multiple convolutional filter layers, the aforementioned relative error should be evaluated in conjunction with probability distribution functions over the inputs. The expression can be updated as follows:

$$E(g,f) = \int_{\epsilon} \int_{\eta} \left| \frac{F(\epsilon,\eta) - G(\epsilon,\eta)}{F(\epsilon,\eta)} \right| \mathscr{P}_x(\epsilon)\mathscr{P}_w(\eta) d(\epsilon) d(\eta) \tag{5.2}$$

Minimal Multiplication in Convolution: Within this formulation, $\mathscr{P}_x(x)$ and $\mathscr{P}_w(w)$ are interpreted as the probability density functions of X and W. The primary aim in implementing approximation is to minimize $E(g,f)$. Convolution, a core operation in the architecture, is expressed as $Z = X \circledast W$, where X denotes the input image and W represents the kernel. The convolution output Z can be specified as:

$$z_{m,n} = \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} x_{m+i,n+j} \cdot w_{i,j} \tag{5.3}$$

Here, (m,n) indicate the pixel positions, while k_h and k_w represent the kernel's height and width. As indicated, computational complexity results from repeated multiplications. An approximate form can be written as:

$$z_{m,n} \approx \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} \mu_{|\hat{w}|} \cdot \min(x_{m+i,n+j}, \hat{w}_{i,j}) \tag{5.4}$$

In this context, $\mu_{|\hat{w}|}$ is the expected value of $|\hat{w}|$. This approximation has been applied to the convolution operation. Approximation at the convolution layer is motivated by its tendency to require the most computation on GPUs and in memory during inference. Before proceeding to model training and result analysis, a further optimization using the signum operation for small-scale and resource-constrained hardware is introduced.

Optimization with the Signum Function: An optimization to the previously introduced minimum multiplicative approach involves the application of signum functions combined with bit-shifting, in order to further reduce the computational burden, particularly for multiplication in convolution. An optimized convolution formulation is described as:

$$z_{m,n} \approx \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} \operatorname{sgn}(x_{m+i,n+j}) \times \operatorname{sgn}(\hat{w}_{i,j}) \times 2^{\text{shift}} \tag{5.5}$$

- **Signum Multiplication:** The signum function $\operatorname{sgn}(x)$ returns the sign of a real value: it outputs $+1$ for $x > 0$, -1 for $x < 0$, and 0 for $x = 0$. For optimizations, $\operatorname{sgn}(\cdot)$ is applied to both the input pixel $x_{m+i,n+j}$ and the kernel weight $\hat{w}_{i,j}$, allowing the sign of their product to be computed using simple comparisons and reducing computational overhead.

- **Bit-shifting:** The term 2^{shift} represents bit-shifting, with "shift" being a parameter optimized during training. This approach achieves scaling with low computational effort.

Through this technique, the convolution operation is reduced to sign comparisons and bit-shifting, significantly decreasing computational complexity. Traditional multiplication is replaced with sign operations and shift computations. Implementation of this method requires adaptation during training, such as the use of gradient clipping, to accommodate the coarser approximations. The main compromise in this method is the possible reduction in model accuracy resulting from higher quantization and approximation. Such an approach is most suitable where speed and efficiency are prioritized over precision, such as in embedded or mobile devices where computational and memory resources are highly restricted.

5.1.1. Convolution Approximation in HD Map Applications

High-definition (HD) maps provide detailed street information of the three-dimensional scenarios, including overlays of traffic signs, speed limits, roadside attributes, and lane markings, thus enabling precise localization for vehicles. Due to their semantic attributes, HD or 3D maps are often considered under the concept of “Mobility as a Service”. Multiple versions of HD and 3D maps have been developed in recent years [12]. Such maps integrate geometric and semantic details acquired from surrounding vehicles using LiDAR, camera, GPS, and IMU data, which are then annotated for the purposes of map generation and enhancement with AI algorithms. Aggregation of information from connected vehicles has been proposed for constructing HD maps, particularly in highway scenarios and use cases such as vehicle platooning. Future automated vehicles will require high-precision localization, both within previously mapped environments and in scenarios beyond line-of-sight [53]. While exact localization is less critical in human-driven vehicles, for fully autonomous and connected vehicles, high-precision and frequently updated maps are necessary. The growing presence of connected vehicles results in large volumes of data, necessitating efficient onboard processing and communication between vehicles and infrastructure within the vehicular network. As a result, the development of efficient data processing pipelines and energy-aware strategies is required to manage the data volume and services related to HD maps [6, 53].

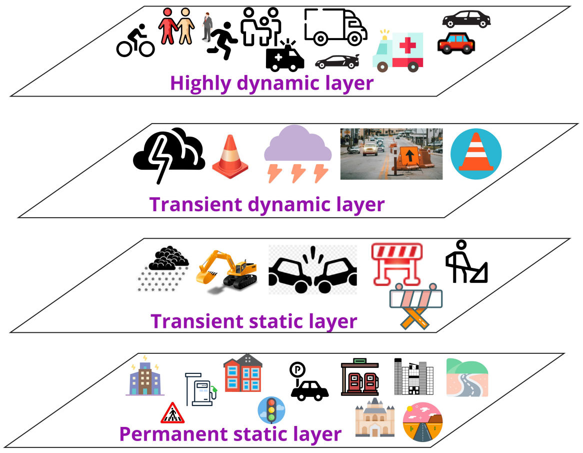


Figure 5.2: HD map layers within CAV (Figure adapted from [40]).

Definitions of high-definition (HD) maps may differ according to the organization, as the data types and annotation procedures depend on the respective data collection strategies. The operational characteristics of HD maps for future connected vehicles, including the expected data and network load according to current sensor data rates, have been outlined by the automotive edge computing consortium (AECC) [53].

High-Definition Map Application: Currently, no unified standard exists for the content or data that must be included in HD maps. AECC provides a widely accepted definition based on multiple application scenarios. AECC description of HD maps [53] includes both static and dynamic elements, organized into four layers according to their respective update intervals (see Figure 5.2). This layer-based approach is influenced by the Local Dynamic Map concept, as standardized by the European Telecommunications Standards Institute (ETSI). The layers are described as follows:

Permanent Static Layer: This layer contains information that changes infrequently, typically at intervals of a day or longer. Data such as traffic lanes, signals, buildings, and the three-dimensional road scene are included here. This layer serves as the static foundation of the map.

Transient Static Layer: Changes in this layer occur over periods ranging from a few hours to several hours. Typical examples include updates for road construction, accident locations, and weather-related changes such as snowfall, as depicted in Figure 5.2.

Transient Dynamic Layer: Information within this layer is updated more frequently, sometimes every few minutes. This includes local weather effects, such as heavy rain, as well as temporary road obstacles or unexpected objects.

Highly Dynamic Layer: This layer is reserved for information that must be updated within intervals from several seconds to a few minutes. It includes the position of moving entities, such as vehicles, bicycles, motorcycles, and pedestrians, and plays a key role in precise vehicle localization. Information requiring sub-second updates in HD maps is not included in this section.

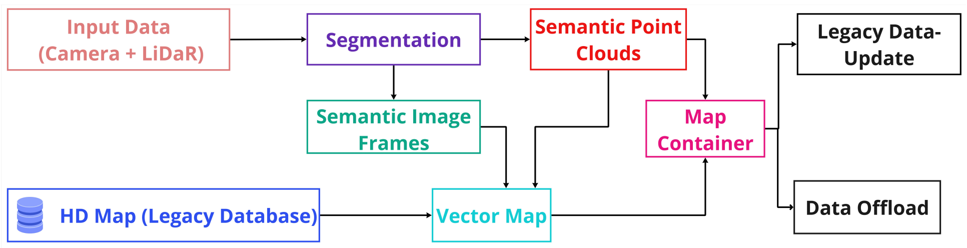


Figure 5.3: HD map update pipeline for vehicular applications

5.1.2. Model Training

The baseline model (as shown in Figure 5.1) is trained on a feature-rich dataset, such as Argoverse [137]. The resulting model file size reaches approximately 134.7 MB, as indicated in Table 1. The deployment of this model on resource-limited devices introduces several computational challenges. Five versions of the baseline architecture have been trained using the PyTorch framework on high-performance computing clusters, where the convolution layer operates with the proposed approximate methods. Model evaluation metrics, including recall, accuracy, and intersection-over-union (IoU), have been monitored during both training and validation phases. On the semantic vector map validation set, the baseline implementation achieves an accuracy of 88.05, a recall of 85.27, and an IoU of 0.86. In addition, a 10% approximation has been applied to the baseline model, resulting in a model size of 120.6 MB, with corresponding recall, accuracy, and IoU of 82.71, 84.58, and 0.85, respectively. Approximations of 25% and 35% relative to the baseline are also considered, enabling development of applications where a balance between energy consumption and accuracy is required and some loss in performance is acceptable. For these experiments, the Argoverse dataset serves as the training and testing resource [137]. The dataset is extensive, comprising 200 test scenarios, which necessitates the use of a split strategy for model training and evaluation. Table 1 provides details of the applied approximation ratios for the baseline architecture and the associated performance metrics. Approximation has been implemented progressively, starting at 10% and increasing to 45%, with each model's size (in megabytes) included in the table. The training process involves forward propagation incorporating the approximate convolution function as shown in Equation 4 followed by backward propagation and parameter updates with a learning rate. In this work, the Adam update rule is used with a learning rate set to 0.001.

Table 5.1: DNN model performance at varying approximation ratios. The first row shows the baseline architecture, while the following rows show performance with approximate percentages relative to the baseline (e.g., Approx-10 indicates 10% of the baseline size.)

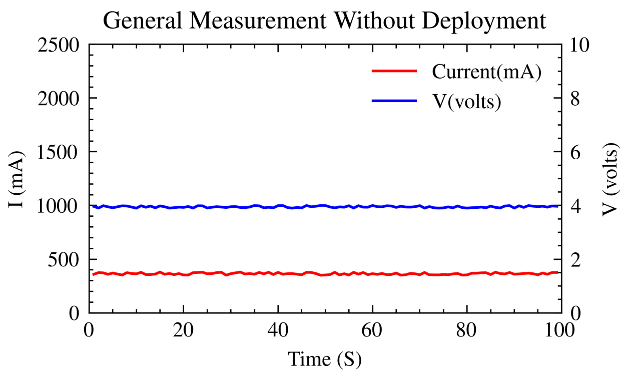
Approximate Ratio/Per	Size (MB)	Recall	Accuracy	IoU
Baseline	134.7	85.27	88.05	0.86
Approx-10	120.6	82.71	84.58	0.85
Approx-20	107.1	78.14	77.03	0.79
Approx-25	100.3	71.04	72.22	0.73
Approx-35	86.8	64.51	65.09	0.61
Approx-45	73.2	55.68	53.66	0.57

5.1.3. Results

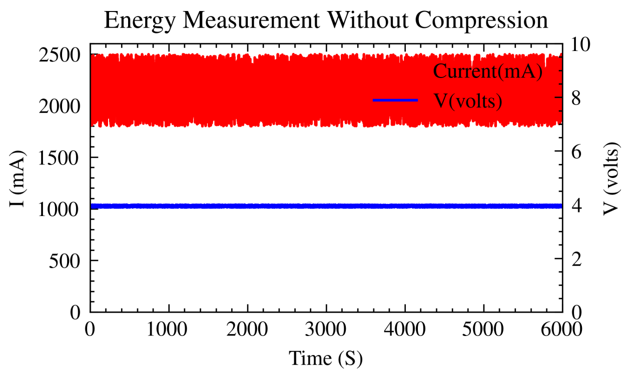
For evaluating energy consumption in the semantic map application, the Approx-35 model referenced in Table 5.1 is used. Current was measured in relation to voltage and time, as shown in Figure 5.4. In the first scenario, shown in Figure 5.4a, the jetson-nano device was monitored in an idle state, with pre-installed applications running in the background and a connected graphics port. Under these default conditions, current readings ranged from 350 to 380 mA over a period of 100 seconds. In the second scenario, the Approx-30 model was evaluated with the Argoverse validation set [137]. Due to the large size of the validation set, measurements were recorded for approximately two hours, resulting in current values between 1820 and 2400 mA. The dataset description notes that these measurements correspond to the size and duration of the validation process. For the third scenario, a lossless data compression strategy was applied to the test data prior to validation by the DNN model. Figure 5.4c shows current values for this configuration, showing a reduction to a range of 1600 to 1880 mA. As expected, the use of data compression leads to a decrease in power consumption. The results discussed here are also part of the published contribution [46].

5.2. Scheme 2: Approximate Multipliers

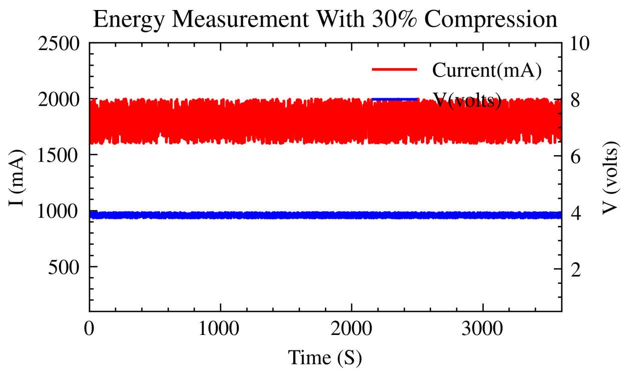
In IoT and AI applications such as object detection and segmentation, approximate multipliers have been applied with different precision levels from 4-bit to 32-bit [27, 141, 142]. While 8-bit multipliers provide efficiency in low-power devices, they reduce accuracy impacting complex DNN models. However, 16-bit multipliers can balance models' accuracy and latency, making them ideal for computer vision tasks [143]. 32-bit multipliers provide the highest accuracy and precision but also increase latency and power consumption [143, 144]. Most of the benchmark models have 32-bit and 16-bit multiplier



(a) Idle power usage.



(b) DNN deployment without compression.



(c) DNN deployment with 30% compression.

Figure 5.4: Energy consumption analysis of the DNN model (Approx-30).

operations. Advances in multi-precision operations have also resulted in DNN models with mixed-precision arithmetic, improving overall efficiency [145, 146].

5.2.1. Software-level Approximation

Deploying computationally complex models like CNNs, DNNs, and ViTs on edge devices requires a tradeoff that can balance model performance and efficiency. Limited computing, memory resources, and battery capability can characterize these edge devices or environments. The computational complexity and energy consumption can be reduced by strategically applying approximate multipliers, variational inference (VI), training-aware quantization, and post-training quantization while maintaining acceptable performance. An overview of the proposed model approximation methodology is shown in Figure 5.5.

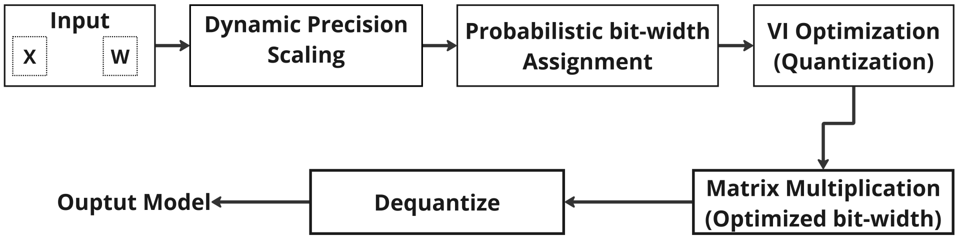

Input
X
W
Dynamic Precision Scaling
Probabilistic bit-width Assignment
VI Optimization (Quantization)
Matrix Multiplication (Optimized bit-width)
Dequantize
Ouptut Model

Figure 5.5: Proposed Methodology for Optimized ViT Training [39]

5.2.2. Algorithms for Model Training with Multipliers

Implementing approximate multipliers within the training process requires an efficient mechanism for reusing partial computation results. To support this, lookup tables are created for 4-bit, 8-bit, and 16-bit approximate multiplications. These tables store the results for all possible input combinations at each precision level and eliminate the need to recompute multiplications during training. This reduces computational overhead and improves runtime efficiency during approximate matrix operations.

To further improve efficiency, we incorporate the Signed Carry Disregard Multiplier (SCDM8) [147, 148], an 8-bit approximate multiplier that omits selected carry operations to lower switching activity within the adder chain. Removing carries reduces energy use while maintaining a product that remains numerically close to the full-precision result. During initialization, each lookup table is populated with all operand pairs: 4-bit values in the range $[0, 15]$, 8-bit values in $[0, 255]$, and 16-bit values in $[0, 65535]$. The approximate product for each pair is computed once and stored for later use.

During multiplication, the algorithm retrieves the appropriate precomputed value from the lookup table according to the selected precision. Basic shortcut rules are applied: if either A or B equals zero, the output is set to zero immediately; if $A = 1$, the value of B is used directly through the identity property. For all other cases, the algorithm uses the stored value corresponding to the chosen precision level.

A central component of the method is the 4×4 approximate multiplier, which serves as the base operation in the 8-bit and 16-bit decompositions. The 4-bit rule approxim-

Algorithm 1 Approximate multiplication with mixed precision

```
Input: Quantized operands $A$ and $B$, precision $p \in \{4, 8, 16, 32\}$
Output: Approximate product $S$
1: procedure ApproximateMultiplication($A, B, p$)
2:     if $p = 32$ then                              ▷ full-precision multiplication
3:         return $A \times B$
4:     else if $p = 16$ then
5:         return $\mathrm{LUT}_{16}[A, B]$
6:     else if $p = 8$ then
7:         return $\mathrm{LUT}_8[A, B]$
8:     else                                          ▷ $p = 4$ (default lowest precision)
9:         return $\mathrm{LUT}_4[A, B]$
10:    end if
11: end procedure
12: procedure BuildLUT($p$)
13:    for $a = 0$ to $2^p - 1$ do
14:        for $b = 0$ to $2^p - 1$ do
15:            $\mathrm{LUT}_p[a, b] \leftarrow \mathrm{ApproxKernel}_p(a, b)$
16:        end for
17:    end for
18: end procedure
```

ates the product of two 4-bit operands using simplified conditional logic instead of full arithmetic. The operands are first classified by magnitude (e.g., whether $B < 8$) and by parity (whether A is even or odd). These conditions determine the resulting approximation. For instance, when both values are small, the output is rounded downsometimes to zeroto reduce computation. When the values are larger, the result is estimated using scaled multiples such as $32 \times (A/2)$ or $B + 32 \times (A - 1)/2$. Although this produces a lossy approximation, especially for low-magnitude inputs, it captures the overall trend of the product while significantly reducing computational cost. Since this 4-bit multiplier forms the basis for larger decompositions (8-bit and 16-bit), it helps achieve consistent energy savings across precision levels.

Mixed-precision quantization is used to reduce the energy cost of matrix operations while keeping model performance within an acceptable range. Instead of assigning a single precision to all layers, different precision levels are explored per layer, so that low-precision arithmetic is used where the model is robust to numerical noise and higher precision is retained where the computations are more sensitive.

To apply this idea in practice, the multiplication primitive in each layer is replaced by the Approximate Multiplication procedure in Algorithm 1. This procedure supports 4-bit, 8-bit, 16-bit, and 32-bit operations using a combination of precomputed lookup tables and approximate multipliers such as SCDM8. Algorithm 2 then searches for an energy-aware precision map across the network. Starting from a fully 32-bit model, the algorithm iteratively proposes lower precisions for the layers that consume the most energy. For each proposal, the network is evaluated on a calibration set to estimate both the new energy usage and the change in loss. If the accuracy drop remains within a predefined tolerance ϵ, the lower precision is accepted; otherwise, the change is reverted and the previous precision is kept for that layer. This process continues until either the global energy usage falls below the budget E or no further safe precision reductions are possible. In this way, Algorithm 2 links the per-operation multiplier in Algorithm 1 to an energy-aware precision selection strategy, ensuring that the final network respects the energy budget while maintaining acceptable detection performance.

Algorithm 2 Energy-Aware Selection of Multiplication Precision

```
Require: Neural network $\mathcal{N}$ with layers $\{L_1, \dots, L_K\}$, energy budget $E$, per-precision energy costs $c(p)$ for $p \in \{4,8,16,32\}$, accuracy tolerance $\epsilon$, calibration dataset $\mathcal{D}_{\text{cal}}$
Ensure: Precision map $P$ and network $\mathcal{N}$ with approximate multiplications
 1: Initialize precision map $P(L_k) \leftarrow 32$ for all layers $k$
 2: Compute baseline loss $L_{\text{base}}$ and energy $E_{\text{base}}$ for $\mathcal{N}$ with 32-bit multiplications
 3: $E_{\text{curr}} \leftarrow E_{\text{base}}$
 4: while $E_{\text{curr}} > E$ and there exists a layer with lower-precision option do
 5:     Select layer $L_j$ with the highest share of total energy
 6:     Propose next lower precision $p'$ for $L_j$ from $\{16, 8, 4\}$
 7:     Temporarily set $P(L_j) \leftarrow p'$
 8:     Replace multiplications in $L_j$ by APPROXIMATE MULTIPLICATION$(A, B, p')$ (Alg. 1)
 9:     Evaluate $\mathcal{N}$ with $P$ on $\mathcal{D}_{\text{cal}}$ and estimate loss $L'$ and energy $E'$
10:     if $L' - L_{\text{base}} \le \epsilon$ then                ▷ Accuracy drop is acceptable
11:         Accept new precision: $E_{\text{curr}} \leftarrow E'$, $L_{\text{base}} \leftarrow L'$
12:     else                                          ▷ Accuracy drop too high, revert layer
13:         Restore previous precision for $L_j$ in $P$ and in $\mathcal{N}$
14:     end if
15: end while
16: return $P$ and $\mathcal{N}$ with approximate multiplications
```

Algorithm 3 aims to reduce the energy consumption of Vision Transformer (ViT) models by assigning different numerical precisions to different parts of the network. In contrast to earlier algorithms that operate at the level of generic matrix multiplications, this algorithm incorporates ViT specific structure such as the arrangement of transformer blocks, attention mechanisms, patch embedding, positional encoding, and MLP sublayers. To make the precision-selection process well defined, the algorithm begins by constructing a mapping from each transformer block to its relative position in the model (input, middle, or output). This information is obtained directly from the architecture definition: the first few blocks are treated as 'input blocks', the last few as 'output blocks', and those in between as 'middle blocks'. Each position is associated with a base precision chosen according to empirical observations that early and late layers are more sensitive to quantization error than mid-layers.

Within each block, the algorithm iterates over both of its main componentsthe self-attention module and the feed-forward network (FFN). Precision choices for these components follow established robustness findings in ViTs: queries and keys tolerate more aggressive quantization, value matrices require slightly higher precision for stable gradi-

Algorithm 3 Energy-Efficient Precision Reduction for ViTs

```
Require: ViT model $\mathcal{N}$ with transformer blocks $\{B_1, \ldots, B_M\}$ and layers $\{L_1, \ldots, L_K\}$, energy budget $E$, per-precision energy costs $c(p)$ for $p \in \{4, 8, 16, 32\}$
Ensure: Optimized ViT with mixed-precision weights
 1: Initialize global energy usage $E_{\text{curr}} \leftarrow 0$
 2: for each block $B_i$ do
 3:     Determine block position (input / middle / output) from index $i$
 4:     Set base precision accordingly (e.g., 16-bit for input/output, 8-bit for middle)
 5:     for each component $C$ in $\{\text{Attention}, \text{MLP}\}$ do
 6:         if $C$ is Attention then
 7:             Assign 8-bit to Q,K; 16-bit to V
 8:             Assign 4-bit to attention-score intermediates
 9:             Apply block-wise quantization
10:         else if $C$ is MLP then
11:             if $C$ is the first FFN layer then
12:                 Assign 16-bit precision
13:             else
14:                 Assign 8-bit with stochastic rounding
15:             end if
16:         end if
17:         for each matrix multiplication $(A, B)$ in layer $L_j$ belonging to $C$ do
18:             if $A = 0$ or $B = 0$ or IsPatchEmbed($L_j$) or IsPosEncode($L_j$) then
19:                 Apply predefined safe precision (typically 16 or 32-bit)
20:             else
21:                 Apply approximate multiplication based on assigned precision $p$
22:             end if
23:             $E_{\text{curr}} \leftarrow E_{\text{curr}} + c(p)$
24:             if $E_{\text{curr}} > E$ then
25:                 Revert this operation to next-higher precision
26:                 Update $E_{\text{curr}}$ accordingly
27:             end if
28:         end for
29:     end for
30: end for
31: Apply structured sparsity to attention matrices
32: Prune redundant heads using attention entropy
33: Apply LayerScale calibration
34: Validate the model on vision metrics
35: return Optimized ViT
```

ents, and intermediate attention-score computations can use very low precision. For the FFN, the first linear layer is kept at higher precision because it expands the feature dimension, while the second linear layer can be quantized more aggressively. During the processing of each matrix multiplication, the algorithm must know *which layer* the operation belongs to. This information is obtained from the model graph: every tensor in a ViT carries a reference to the module or block that produced it (e.g., patch embedding layer, positional encoding layer, attention projection, or FFN sublayer). Helper functions such as IsPatchEmbed(layer) and IsPosEncode(layer) check whether the matrix operation originates from sensitive layers that should not use aggressive quantization. Energy usage is tracked cumulatively. After each operation, the cost of executing the chosen precision is added to the running total. The energy cost for each precision level (4-, 8-, 16-, 32-bit) is defined beforehand using either hardware measurements, analytical models, or published lookup tables. If the cumulative energy exceeds the budget, the algorithm locally reverts the decision for that operation and selects a higher precision for stability.

After the per-operation precision assignment, the algorithm introduces ViT-specific refinements: structured sparsity pruning in attention matrices, head-pruning based on attention entropy, and LayerScale calibration to stabilize training when mixed precision is applied. The final model is then validated using vision benchmarks to ensure that accuracy remains within acceptable limits while meeting the defined energy budget.

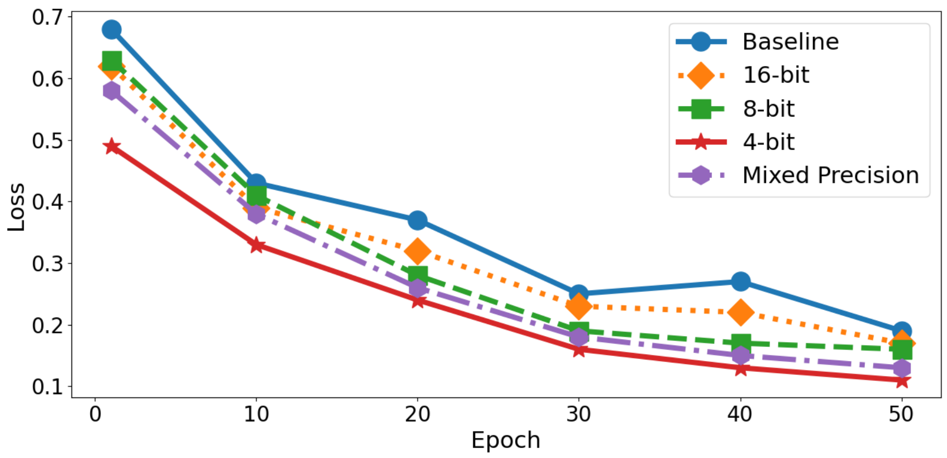


Figure 5.6: Bit-wise comparison of training loss over epoch

5.2.3. Evaluation of Multipliers

Figure 5.6 presents the loss values of the ResNet18 model across training epochs for different precision settings. The baseline model trained with 32-bit floating point multipliers begins with the highest loss and reaches a stable region near epochs 40–50. The 16-bit and 8-bit versions follow a similar pattern and show a similar loss trend throughout training. The 4-bit curve starts below the other fixed precision settings and appears to decrease faster in the early epochs. This does not indicate that 4-bit precision is better performing; rather, the higher amount of quantization noise at 4 bits can occasionally act as a regularising factor during the early phase of training. In later epochs, the 4-bit and 8-bit have similar training losses, showing that the noise level at such low precision limits how much the loss can improve without additional adjustments.

The mixed-precision setting remains more stable across epochs. It benefits from using higher precision in sensitive layers and lower precision in layers that can handle quantisation noise. As a result, its curve remains within the range formed by the 4-bit and 8-bit plots but avoids the fluctuations seen in fixed low-precision settings. This explains why the mixed configuration reaches lower loss values while keeping the training process stable. The behaviour in this example may vary for other models and datasets. In practice, the best configuration often depends on identifying layers that can tolerate reduced precision and lowering their cost while keeping the rest of the network stable.

5.3. Scheme 3: Variational Inference

Another approximation approach used is the variational inference, which is applied to more complex models, such as the vision transformer, to model the uncertainty in the network weights. Variational inference approximates the true posterior distribution of weights by optimizing an approximate distribution. This is important in the context of Vision Transformers (e.g., ViT B32), where it aids in estimating uncertainty linked to model predictions. The weights in Vision Transformers are treated as random variables rather than fixed values. The objective is to closely approximate the posterior distribution of these weights based on observed data. In this strategy, we apply probabilistic approximation based on occam's razor to vision transformers with the goal of reducing computational complexity and improving speedup [149]. The general goal of variational inference is to approximate the true posterior distribution of weights by optimizing an approximate distribution and focusing on estimating uncertainty linked to model predictions. We reformulate this objective to prioritize computational efficiency using occam's razor-inspired loss. This approach balances model fit and complexity, directly addressing the trade-off between performance and efficiency in Vision Transformers. The weights in Vision Transformers are treated as random variables rather than fixed values. The objective is to closely approximate the posterior distribution of these weights based on observed data while penalizing excessive model capacity. Instead of traditional variational inference approach which is based on the Kullback-Leibler (KL) divergence and maximizing the Evidence Lower Bound (ELBO), in our proposed strategy, we optimize the objective as follows:

$$\mathcal{O} = -\log p(X|\hat{\theta}) + \log \int_{\mathcal{M}} \kappa(\theta) \exp\left(-\frac{N}{2}(\theta-\hat{\theta})^\top J(\hat{\theta})(\theta-\hat{\theta})\right) d\theta \tag{5.6}$$

Here X is the input data, $\hat{\theta}$ is the maximum likelihood estimate of the weights, and $J(\hat{\theta})$ is the Fisher Information Matrix, which approximates the curvature of the log-likelihood around $\hat{\theta}$. The $\kappa(\theta)$ is the capacity of the model, penalizing over-parameterization and ensuring efficient model representation. This objective ensures that the model achieves a balance between fitting the data and maintaining a compact representation. By optimizing $\mathcal{O}$ the model reduces computational complexity and improves robust performance in tasks, such as object detection, segmentation, or classification. Using the strategy, we propose model training process for object detection enhanced with post-training bit quantization for improved computational efficiency:

Algorithm 4 Variational Inference with Post-Training Bit Quantization (VIPT: Occam-based)

```
Require: Training dataset $D$, learning rate $\alpha$, epochs $T$, batch size $B$
Ensure: Trained and quantized object detection model $M$
 1: Initialize variational distribution $q(w)$ (mean $\mu$, scale $\sigma$ for each weight)
 2: Define optimizer for $(\mu, \sigma)$ with learning rate $\alpha$
 3: Set loss function $\mathscr{L}_{Occam}$ from Eq. (4)
 4: for $t = 1$ to $T$ do
 5:     Shuffle $D$
 6:     $\mathscr{L}_{total} \leftarrow 0$
 7:     for $i = 0$ to $|D|$ step $B$ do
 8:         Sample weights $w \sim q(w)$
 9:         Compute batch loss $\mathscr{L} = \mathscr{L}_{Occam}(M(w), D[i : i + B])$
10:         Backpropagate through model and variational parameters
11:         Update $(\mu, \sigma)$ using optimizer
12:         $\mathscr{L}_{total} \leftarrow \mathscr{L}_{total} + \mathscr{L}$
13:     end for
14:     $\mathscr{L}_{avg} = \mathscr{L}_{total} / (|D|/B)$
15:     Display $\mathscr{L}_{avg}$
16: end for
17: Quantize trained weights to 8-bit using uniform quantizer
18: Replace floating-point multipliers with 8-bit approximate multipliers
19: Save quantized model $M$
```

Algorithm 4 combines variational inference with an Occam-inspired loss and post-training quantization. The training process begins by placing a variational distribution $q(w)$ over the network weights. In practice, this distribution is commonly chosen as a diagonal Gaussian parameterized by a mean and a learned scale, which allows weight samples to be drawn during training. Each forward pass uses one sample from $q(w)$, enabling the model to capture uncertainty while keeping the update rule simple. During each batch, the loss $\mathscr{L}_{Occam}$ is computed using the sampled weights. This loss includes a standard detection objective together with a term that penalizes model complexity. The additional term encourages a compact set of parameters by discouraging solutions that rely on large or unstable weights. After computing the batch loss, gradients are propagated through both the model parameters and the variational parameters. The optimizer then updates the mean and scale of $q(w)$, gradually shaping the distribution toward values that fit the training data while remaining compact. The training loop accumulates the loss across batches to monitor the average epoch loss. Once the learning phase is complete, the model parameters are converted to an 8-bit format using a uniform quantizer. This step reduces the memory footprint and the number of bit operations during inference. After quantization, floating point multipliers are replaced by their 8-bit approximate counterparts. This replacement reduces computation cost while keeping the model output within an acceptable range, as verified in the evaluation stage.

This workflow separates model learning from the precision reduction step. The variational phase focuses on stable training with regularization, while the quantization phase focuses on lowering resource usage. The combined method produces a compact detector that remains suitable for edge execution while keeping its predictive ability within the target range.

As an alternative approach and to have a detailed comparison with energy-aware training methods, we propose a strategy to optimize energy usage during the training of object detection models. This approach integrates probabilistic energy-efficient training with an Occam's Razor inspired objective to reduce energy consumption while maintaining model accuracy. The method incorporates a customized loss function that balances model fit and complexity, penalizing excess capacity using the Fisher Information Matrix $J(\theta)$. The training process uses a variational distribution $q(w)$, representing the probabilistic distribution of model weights. Unlike traditional energy-aware methods, this approach dynamically adjusts model parameters and energy consumption metrics to balance performance and computational cost. The proposed loss function for energy-aware training is:

$$\mathscr{L}_{Occam-Energy} = -\log p(X|w) + \frac{\lambda}{2} w^\top J(w) w + \gamma EE_{stats} \tag{5.7}$$

Here $\log p(X|w)$ represents the data likelihood, $J(w)$ is the Fisher Information Matrix approximating the curvature of the log-likelihood, λ controls the regularization strength on the model weights, and γ scales the energy consumption penalty EE_{stats}. Algorithm 5 covers the model training process, integrating real-time energy monitoring and loss-based parameter updates. The algorithm manages energy consumption through an energy statistics variable, EE_{stats}, which allows for dynamic adjustments to training parameters. This ensures that the model operates within predefined energy budgets. If energy usage exceeds the threshold E_{max}, the learning rate is reduced to minimize computational overhead while sustaining effective training progress. By incorporating the Fisher Information Matrix into the loss function, the model discourages over-complexity, promoting efficient resource utilization without compromising generalization. The energy statistics variable, EE_{stats}, is computed using real-time energy profiling, ensuring dynamic adjustments during training. Both regularization strength λ and energy penalty coefficient γ were optimized through a grid search over a validation set. **Regularization Strength (λ)**: Controls the impact of the regularization term. Smaller λ values allowed higher model capacity but increased the risk of overfitting, whereas larger values constrained the model, improving generalization at the cost of training accuracy. **Energy Penalty Coefficient (γ)**: determines the weight of the energy penalty. Higher γ values effectively reduced energy consumption but introduced minor accuracy degradation (typically within 1–2% mAP). Lower values balanced accuracy and energy use, but energy consumption was higher.

By fine-tuning λ and γ, model achieves a balance between energy and performance. For example, models trained with higher γ show higher energy savings but also show a reduction in the model's accuracy. Regularization through λ improved generalization while maintaining convergence stability. These trade-offs shows the flexibility of the proposed framework in adapting to different performance and resource constraints, making it suitable for real-world edge applications. These statistics are measured using a energy monitoring tool such as nvml or tegrastats, providing accurate feedback on power consumption for each training batch. The custom loss function, $\mathscr{L}_{Occam-Energy}$, integrates this information to penalize excessive energy usage while maintaining a balance between model complexity and fit. After training, the model can be further optimized through quantization, as discussed in the algorithm 4, ensuring that both training and inference are computationally efficient on edge devices.

Algorithm 5 Variational Inference-Based Energy-Efficient Training (VIET: Occam's Razor-Inspired)

```
Require: Training dataset $D$, learning rate $\alpha$, epochs $T$, batch size $b$, energy budget $E_{max}$
Ensure: Trained model $M$, Energy Stats $EE$
 1: Initialize variational distribution $q(w)$
 2: Set optimizer with learning rate $\alpha$
 3: Define custom loss function $\mathscr{L}_{Occam-Energy}$
 4: for $t = 1$ to $T$ do
 5:     Shuffle $D$
 6:     $\mathscr{L}_{total} \leftarrow 0$, $EE_{total} \leftarrow 0$
 7:     for $start = 0$ to len($D$) step $b$ do
 8:         Sample weights $w$ from $q(w)$
 9:         Compute loss $\mathscr{L}_{Occam-Energy}(w)$ on batch $D[start : start + b]$
10:         Backpropagate and update weights using $\nabla\mathscr{L}_{Occam-Energy}(w)$
11:         $EE_{batch} \leftarrow$ Compute energy usage for current batch
12:         Update energy statistics: $EE_{total} \leftarrow EE_{total} + EE_{batch}$
13:         $\mathscr{L}_{total} \leftarrow \mathscr{L}_{total} + \mathscr{L}_{Occam-Energy}(w)$
14:     end for
15:     Compute average loss $\mathscr{L}_{avg} = \mathscr{L}_{total}/(\text{len}(D)/b)$
16:     Compute average energy consumption $EE_{avg} = EE_{total}/(\text{len}(D)/b)$
17:     if $EE_{avg} > E_{max}$ then
18:         Update learning rate $\alpha \leftarrow 0.9 \times \alpha$
19:     end if
20:     Print $\mathscr{L}_{avg}$, $EE_{avg}$
21: end for
22: return $M$, $EE$
```

5.3.1. Look-up-table and Loss Function

The variational inference setup for object detection on the nuScenes dataset emphasizes the use of a specialized loss function mentioned below and the fine-tuning of hyperparameters. The proposed approximation scheme includes two mechanisms: *VIET* (Variational Inference-Based Energy-Efficient Training) and *VIPT* (Variational Inference with Post-Training Bit Quantization). VIET implemented the customized loss function $\mathscr{L}_{Occam}$ (Equation 5.8) to balance classification and regression losses with a capacity penalty, thereby preventing over-parameterization. This approach utilized variational inference to model weight distributions and integrated energy-aware training to optimize resource utilization. Subsequently, VIPT applied 8-bit quantization post-training, as outlined in Algorithm 4, to further reduce memory usage and computational demands. This dual mechanism ensured the model achieved compression and energy efficiency without compromising detection performance.

Table 5.2: Lookup Table for Activation Outputs in VI Strategy

Normalized Input Range	**Probabilistic Output Distribution**
$-1.0 \le x < -0.8$	$\mathcal{N}(-0.9, 0.05)$
$-0.8 \le x < -0.6$	$\mathcal{N}(-0.7, 0.05)$
$-0.6 \le x < -0.4$	$\mathcal{N}(-0.5, 0.05)$
$-0.4 \le x < -0.2$	$\mathcal{N}(-0.3, 0.05)$
$-0.2 \le x < 0$	$\mathcal{N}(-0.1, 0.05)$
$0 \le x < 0.2$	$\mathcal{N}(0.1, 0.05)$
$0.2 \le x < 0.4$	$\mathcal{N}(0.3, 0.05)$
$0.4 \le x < 0.6$	$\mathcal{N}(0.5, 0.05)$
$0.6 \le x < 0.8$	$\mathcal{N}(0.7, 0.05)$
$0.8 \le x \le 1.0$	$\mathcal{N}(0.9, 0.05)$

Table 5.2 describes the probabilistic distributions with predefined ranges of normalized input values. Each range maps to a normal distribution defined by a mean (μ) and variance (σ^2), where the mean aligns with the midpoint of the input range. This mapping is particularly useful for models where activations are expected to vary within known limits based on the input they process. Using this table in the VIET and VIPT strategy, the models can utilize these predefined distributions to approximate neuron activations without recalculating the distributions for each input during runtime. This improves processing speed and optimizes energy usage, making the system ideal for deployment in environments with limited power and computational resources. For the VIPT strategy, this table guides the quantization process by suggesting which activations need higher precision based on their variance, thus preserving model accuracy. Utilizing this lookup table provides an efficient approach to balancing computational demands with the accuracy requirements of advanced neural network architectures. During model training, the overall training loss function used for the variational inference strategies, VIET (Variational Inference-Based Energy-Efficient Training) and VIPT (Variational Inference with Post-Training Bit Quantization) addresses classification accuracy, regression precision and model complexity while optimizing for energy efficiency, which complements the loss functions discussed in equation 5.7. The total loss function is described as:

$$\mathcal{L}_{Occam}(y, t, \mu, \log(\sigma^2)) = \alpha \cdot \mathrm{CL}(y_{\mathrm{cls}}, t_{\mathrm{cls}}) + \beta \cdot \mathrm{RL}(y_{\mathrm{reg}}, t_{\mathrm{reg}}) + \lambda \cdot \mathrm{CPenalty}(\mu, J(\mu)) \quad (5.8)$$

Here $\mathrm{CL}(y_{\mathrm{cls}}, t_{\mathrm{cls}})$ is the classification loss (cross-entropy). $\mathrm{RL}(y_{\mathrm{reg}}, t_{\mathrm{reg}})$ is the regression loss (smooth L_1 loss). $\mathrm{CPenalty}(\mu, J(\mu))$ penalizes over-parameterization using the Fisher Information Matrix $J(\mu)$. The hyperparameters α, β, and λ are fine-tuned to balance these components, ensuring that the model achieves high accuracy while conserving energy. The quantitative results through proposed methods are discussed in the following section.

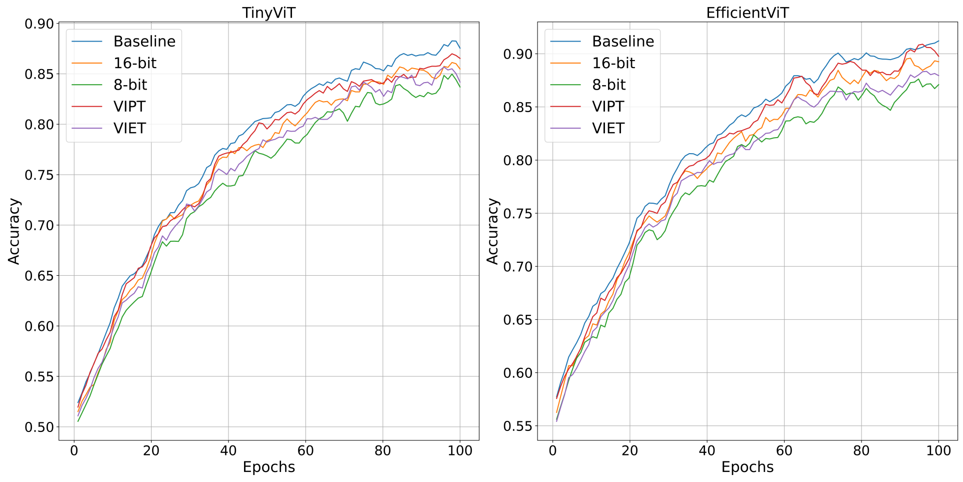


Figure 5.7: ViTs model training accuracy using proposed approach

5.3.2. Training and Inference Results

DNN models and vision transformers were trained for detailed analysis using the methods discussed previously. The TinyViT and EfficientViT models were used as a baseline from the vision transformer family. Since the second strategy (approximate multiplication for convolutions) can generally be used with the DNNs or vision transformers, primarily consisting of convolutional layers, the discussed approximation strategies were used interchangeably during training and inference for evaluation and analysis. Figure 5.7 shows the training accuracy of two models, TinyViT and EfficientViT, for 100 epochs under different precision settings: Baseline (FP32), 16-bit, 8-bit, VIPT, and VIET. For both models, the Baseline configuration consistently shows the highest accuracy throughout the training; the 16-bit and 8-bit precision reduction maintain or are within the baseline range, with the 16-bit maintaining higher accuracy than the 8-bit multiplier, showing the expected trade-off between computational efficiency and accuracy due to reduced precision. VIPT and VIET strategies show a balanced tradeoff by optimizing model performance and computational resources. While these models have reduced accuracy compared to the baseline, they have a similar accuracy trend to 8-bit and 16-bit multipliers. This pattern suggests that these methods, involving variable or targeted precision techniques, offer a balanced approach for deployment on non-specialized hardware or edge devices.

Table 5.3a shows precision metrics for vision models on a subset of the nuScenes dataset. Analyzed models include ResNet18, ResNet34, TinyViT, and EfficientViT under different conditions such as baseline, 8-bit and 16-bit, Ax-Conv (approximate multiplication for convolution) and methods applying variational inference like VIPT, and VIET. Precision values show transformer models (TinyViT and EfficientViT) perform better than ResNet models in baseline settings. When precision is reduced to 8-bit, all models show a drop in precision, with the least impact on EfficientViT. The 16-bit precision provides

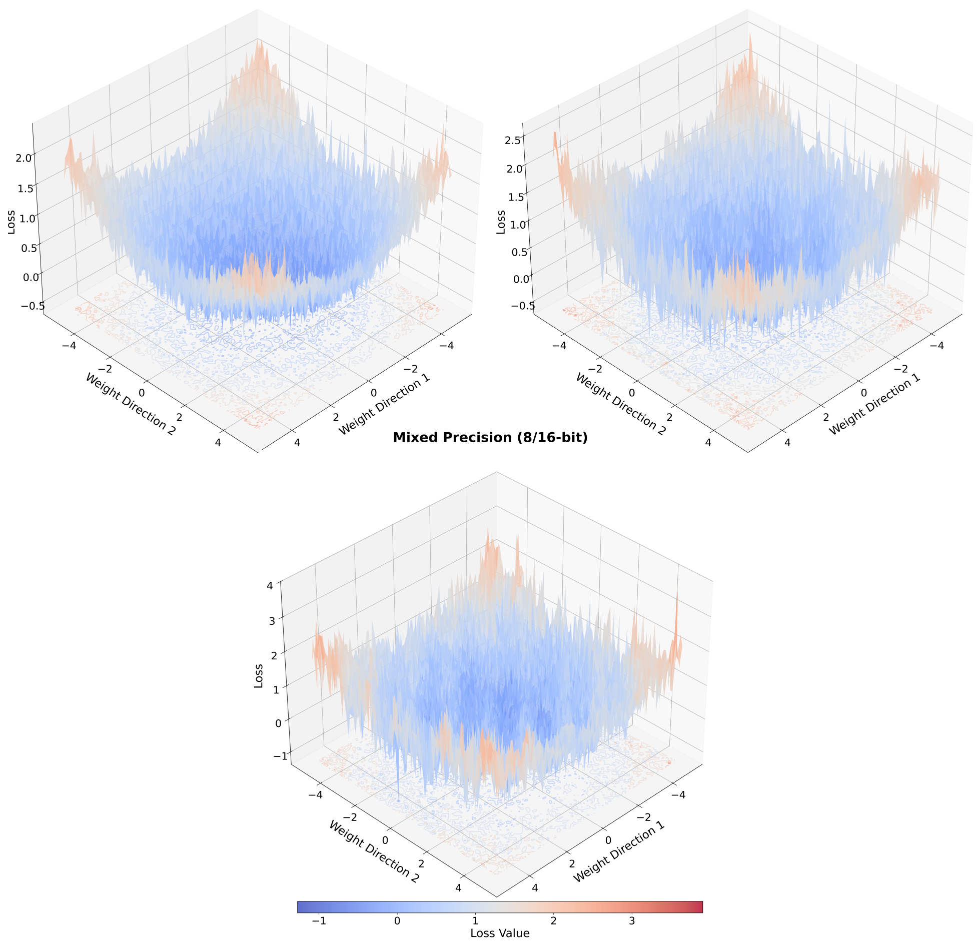


Figure 5.8: EfficientViT models (baseline, VIPT, VIET) training loss on the nuScenes data

better results than 8-bit, showing less data loss. Ax-Conv performs better than both 8-bit and 16-bit settings by preserving more accuracy for ResNet models; however, this behaviour can be attributed to the heavy matrix multiplication and convolution operation. VIPT and VIET show results close to baseline, suggesting effectiveness in maintaining precision during quantization and training. Table 5.3b compares model performance based on parameters, FLOPs, MACs, and latency metrics. Model complexities vary significantly, from TinyViT-2's lower parameter count to EfficientViT-1's higher count. ResNet34 shows the highest FLOPs, whereas TinyViT-2 is the most compute-efficient. MACs correlate with FLOPs across all models, with TinyViT-1 having the highest MACs value. For model inference overview, we measure the above-mentioned metrics and latency on the Jetson Xavier NX with a range from 14 ms for ResNet18 to 39 ms for EfficientViT-

1, showing that higher model complexities do not necessarily correspond to increased latency, suggesting optimizations in architecture and further possibility of hardware-aware optimization.

Table 5.3: Results on nuScenes subset using approximate multipliers and VI.

(a) Precision results

	ML Models for object detection tasks			
Method	***ResNet18***	***ResNet34***	***TinyViT***	***EfficientViT***
Baseline	34.3	39.1	52.7	55.9
8-bit	22.7	29.5	44.1	49.9
16-bit	24.3	34.7	49.5	51.8
Ax-Conv	29.1	30.8	-	-
VIPT	33.9	38.2	51.3	54.7
VIET	33.4	37.4	49.7	52.1

[a]Ax-Conv: Variational inference on convolution operations
[b]VIPT: Variational inference with post-training quantization
[c]VIET: Variational inference with energy-aware training

(b) Model Performance

	Metrics Measurements			
Model	***Param***	***Flops (G)***	***MACs***	***Latency (ms)***
ResNet18	11.7	1.8	1.9	14
ResNet34	21.8	3.7	3.1	26
TinyViT-1	25.4	2.0	4.6	21
TinyViT-2	8.1	0.5	2.1	17
EfficientViT-1	40	1.5	3.4	39
EfficientViT-2	13.4	1	2.0	23

[a]Latency: Tested over Jetson Xavier NX
[b]Model-1: Trained using VIPT
[c]Model-2: Trained using VIET

Figure 5.8 shows the loss maps of the EfficientViT model under three different training strategies: baseline, Variational Inference with Post-Training Quantization (VIPT), and Variational Inference with Energy-Aware Training (VIET). The left plot is the baseline model, showing the loss landscape with few variations and peaks. These loss patterns are standard across optimized neural networks such as EfficientViT, showing areas where the model parameters adjust to minimize the loss effectively, with a balance between parameter stability and sensitivity. The middle plot shows the model trained using the VIPT strategy; the loss shows variation as compared to the baseline model. The plot also shows the effect of quantization as it stabilizes the loss landscape, though with a slight increase in peak values. This trend shows that while quantization helps in general stabilization, it also restricts the model's ability to reach the lowest possible loss values within the quantized parameter space. The right plot shows the loss landscape from the VIET training approach. Compared to the baseline and VIPT, this landscape has higher peaks and loss values, showing the challenges in achieving optimization. This trend or pattern can be associated with the trade-offs made for energy efficiency, where adjustments to model parameters or operational precision are made at the runtime cost of smooth optimization. Overall, the visualized loss landscapes in figure 5.8 provide insight into how each training method affects the model's optimization dynamics. VIPT provides a balance by smoothing out sharp gradients, whereas VIET introduces complexities that prevent loss minimization. Additional strategies such as adjusting regularization, exploring other energy-efficient modifications, or adapting learning rates might be necessary to optimize training further under VIET strategy.

Figure 5.9 compares the weight matrices of the EfficientViT model under three configurations: baseline, 16-bit quantization, and mixed precision using the first proposed approximation strategy (approximate multipliers: Algorithm 3). The attention layer shows minimal visual change with different precision levels, maintaining a clear diagonal pattern which signifies self-attention, with a little granularity under reduced precision. In the MLP layer, the transition from full to lower precision levels shows a more pronounced pixelation, showing the impact of the quantization process, with information loss increasing in mixed precision. The embedding layer has the most differences, where lower precision results in visibly coarser gradients, affecting the representation of spatial correlations within the weights. Information loss is also higher in mixed precision, confirming a more substantial impact on the layer's pattern. This comparison from full precision through various quantization levels visually shows the impact of precision reduction on weight matrix structure across different model components.

Results

A test and evaluation using multi-metric is covered here on the benchmark models using approximation strategies over nuScenes and Waymo validation set. To analyze inference based gains, the models are deployed for a single inference on Xavier NX to capture energy consumption, model performance metrics and system metrics.

Energy Consumption: When considering the energy consumption of these models on a Xavier NX, we must account for the device's power draw and the time taken to perform a single inference. Using the device specification of Jetson Xavier NX which draws an average power of 15 watts during inference, which is a mid-range estimate for this device, the energy usage for each model can be calculated by multiplying this power with the inference time (latency). The energy consumed E can be:

$$E = P \times t$$

Where P is the power usage in watts, and t is the time in seconds. As shown in Figure 5.11, the ResNet18 model trained using the approximation consumes about 0.21 microjoules of energy per inference with a latency of 14 milliseconds. Similarly, ResNet34, with a longer latency of 26 milliseconds, uses approximately 0.39 microjoules per inference. We observe a difference for TinyViT models (Model-1 is trained using VIPT, and Model-2 is trained using VIET). TinyViT-1, with a 21-millisecond latency, consumes around 0.315 joules, while the more efficient TinyViT-2, with its 17-millisecond latency, only use about 0.255 joules per inference. These results show that even within the same

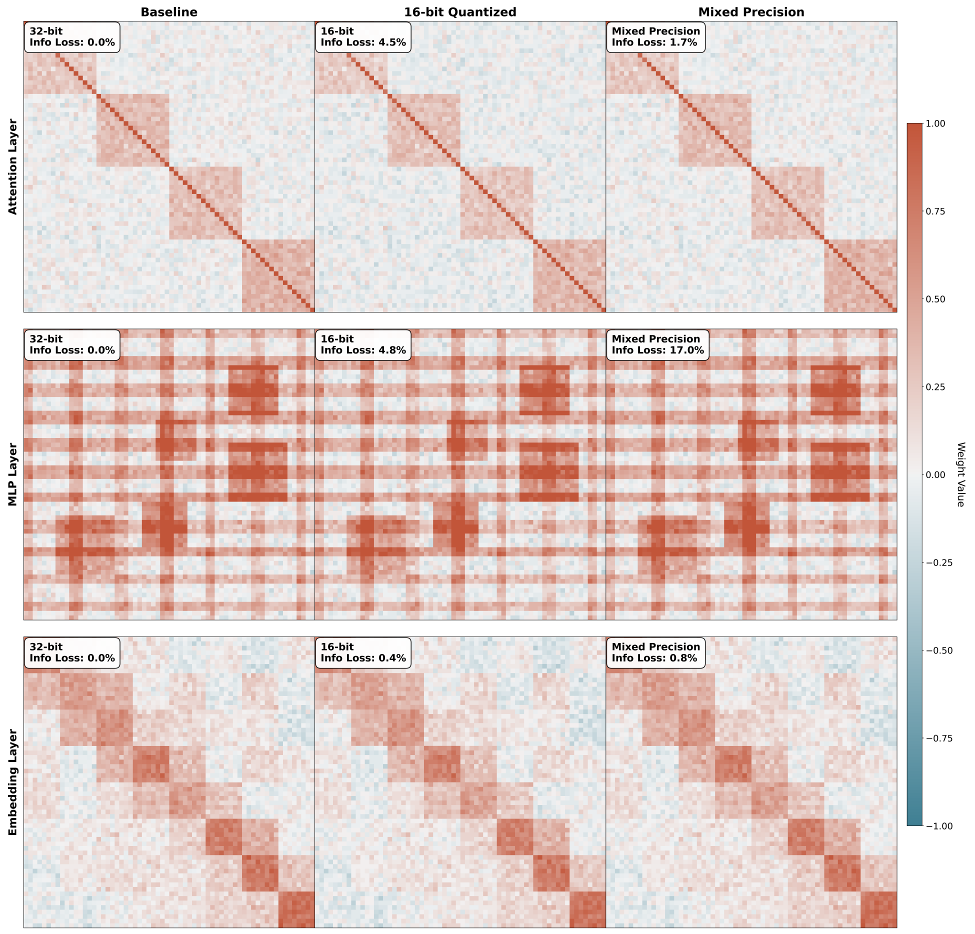


Figure 5.9: EfficientViT weights distribution on different precision levels and layers.

family of models, architectural differences and optimization approaches can lead to on-board energy savings. The EfficientViT-1 shows a higher latency of 39 milliseconds, resulting in an energy consumption of about 0.585 joules per inference, higher than any of the models discussed. The model EfficientViT-2, with a 23-millisecond latency, uses about 0.345 joules, showing a similar trend as the TinyViT-2 model. It's important to note that these energy results are based on static power draw values recorded by nvml and tegrastats manager of the device, while actual energy consumption would fluctuate because of the computational and memory load at any given moment.

Memory Footprint: Figure 5.10 gives an overview of the memory footprint across different models and optimization strategies and shows improvements in memory efficiency through the application of VIPT (Variational Inference with Post-Training Quantization) and VIET (Variational Inference with Energy-Aware Training) strategies. The baseline measurements show the actual memory demands of each model, with Swin-Tiny hav-

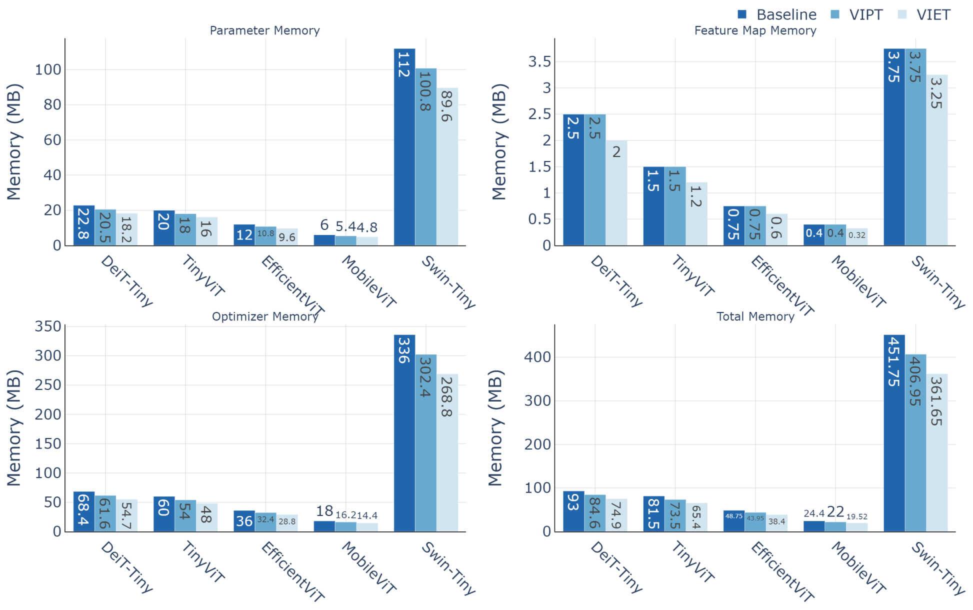


Figure 5.10: Memory comparison of vision transformer models with variational inference strategies

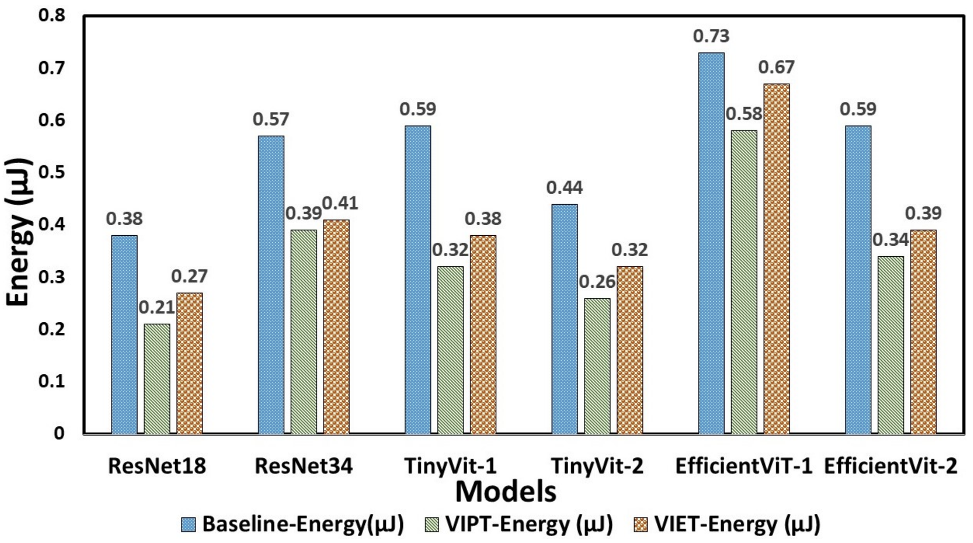


Figure 5.11: Energy usage comparison on Nvidia Xavier NX

ing the highest memory requirement at approximately 452 MB due to its extensive parameter and optimizer requirements. Models like MobileViT have minimal memory usage, around 25 MB, suitable for environments like smartphones or edge devices with strict memory constraints. After applying the VIPT strategy, we observe a reduction in memory usage across all models. For e.g., Swin-Tiny total memory usage decreases from 452 MB to approximately 407 MB. This reduction is primarily driven by optimized memory management in parameter storage and the optimizer's memory footprint, aligning with VIPT's objective to enhance post-training quantization efficiency without architectural changes. The VIET strategy, focused on energy-aware optimizations, further reduces the memory footprint. It achieves higher savings, as seen in Swin-Tiny, reducing the total memory requirement to about 361.9 MB. This strategy optimize parameter storage and also reduces the optimizer memory, making it the first choice for deployment in power-sensitive or memory-constrained environments. Overall, these results show the effectiveness of VIPT and VIET in reducing the memory footprint and enhancing the deployability of models in diverse operational contexts, particularly where memory efficiency is important. Such optimizations facilitate the broader adoption of advanced models like Swin-Tiny and EfficientViT in edge devices, highlighting the practical benefits of these advanced optimization techniques in real-world applications.

Table 5.4: Performance comparison of models [150, 151].

Model	#Para	GFLOPs	Latency (ms)	mIoU(%)
TransFusion	27.8M	38.3	268.2	69.3
DeiT-Ti	5.7M	1.3	125.1	62.4
SPViT-S	15.9M	3.3	182.4	66.2
TinyViT	21.4M	27.0	210.2	69.5
DeiT-S	22.1M	4.6	210.5	68.6
EViT-S	22.1M	3.0	210.8	68.1
Swin-Ti	28.3M	4.5	215.9	68.2
STEP-Swin-S	36.9M	6.3	218.2	67.3
SPViT-Swin-S	38.9M	6.1	212 .4	67.5
MobileViT	38.9M	6.1	212.4	67.8
SPViT-B	41.6M	8.4	222.1	66.1
VTP-B	48.0M	10.0	225.4	67.4
Swin-S	49.6M	8.7	230.1	69.7
EfficientViT	49.0M	9.1	170.4	70.8
DeiT-B	86.4M	17.5	250.1	70.4
EViT-B	86.4M	11.6	252.6	70.1
eTPS-B	86.4M	11.4	248.7	70.7
dTPS-B	87.0M	11.4	241.5	70.4
TinyViT-1	21.4M	35.4	188.2	65.1
EfficientViT-1	25.5M	38.6	212.5	69.0
MobileViT-1	27.9M	39.4	236.9	74.6
TinyViT-2*	16.7M	32.1	145.1	63.8
EfficientViT-2*	19.1M	33.7	155.2	64.6
MobileViT-2*	10.3M	24.4	122.9	72.9

Model Metrics: Table 5.4 provides a detailed performance analysis of several vision transformer models, compared with TinyViT, EfficientViT, and MobileViT trained using VIPT and VIET strategies on nuScenes validation set. The model adapted for training are described by suffixes *Model-1* for VIPT and *Model-2* for VIET. *TinyViT-1* model has 21.4M parameters, utilizing 35.4 GFLOPs, resulting in a latency of 188.2 ms and a mIoU of 65.1%. This model balances processing speed and analytical performance, which is ideal for edge devices. *EfficientViT-1* has a higher resource demand with 25.5M parameters and 38.6 GFLOPs. It also has a higher latency value at 212.5 ms, with an mIoU of 69.0%. The model is suitable for a design choice to enhance image understanding at the expense of higher resource use. *MobileViT-1*, with 27.9M parameters and 39.4 GFLOPs, experiences an increase in latency to 236.9 ms. However, it has a higher accuracy with an mIoU of 74.6%, highlighting its capability to have better model performance. Overall, these models show two approaches to balance the trade-offs between resource demands, processing time, and accuracy, highlighting the effectiveness of VIPT and VIET strategies in optimizing vision transformer architectures for specific application needs.

Table 5.5: Validation Results on Waymo Dataset

	3D mAP			
Method	**Overall**	**0-30m**	**30-50m**	**50m-Inf**
ResNet18	71.5	87.8	69.3	47.3
BevFusion	69.7	91.3	68.5	41.3
FusionFormer	75.7	91.5	74.1	51.3
PointPillar	72.1	88.5	69.9	48.0
MV3D	62.9	86.3	60.0	36.9
Pillar-OD	69.8	88.5	60.5	42.6
PV-RCNN	70.3	91.9	69.2	42.2
CenterNet	76.5	92.0	74.8	53.0
CenterPoint	77.0	92.0	76.8	56.1
MobileViT-1	83.8	86.9	78.1	64.6
MobileViT-2	81.5	83.3	71.4	56.3
Efficient-ViT-1	79.2	92.4	78.2	59.7
Efficient-ViT-2	78.7	92.2	77.5	58.9
TinyViT-1	76.1	91.6	76.4	55.2
TinyVit-2	78.5	92.8	75.3	56.3

Table 5.5 shows the validation results of various benchmark models on the Waymo dataset, focusing on the performance metrics of 3D mean Average Precision (mAP) and 3D mean Average Precision with Heading (mAPH), which are critical for assessing the accuracy of 3D object detection models. Among the traditional DNN models, PointPillar has a benchmark accuracy with a 72.1% overall 3D mAP, showing better results in closer ranges (0-30m) with an 88.5% score but dropping in performance at distances beyond 50m. MV3 shows a 62.9% overall 3D mAP, with a performance decline after the 30m mark. Pillar-OD shows 69.8% overall 3D mAP, comparable close-range performance to PointPillar but reduced effectiveness at medium and long ranges. From our proposed

method, MobileViT-1 shows higher performance with an 83.8% overall 3D mAP, consistent accuracy across all ranges, and the highest long-range (50m-Inf) detection at 64.6%, shows efficiency in detecting objects across varying distances with high precision. The TinyVit-1 model trained using VIPT shows reduction in average precision as compared to any other models on the waymo validation set.

Table 5.6: Inference speedup and energy saving results for ViT models.

Speedup (×)			
TinyViT-1	**TinyViT-2**	**MobileViT-1**	**MobileViT-2**
1.00	1.00	1.00	1.00
1.92	1.96	1.80	1.83
1.62	1.53	1.43	1.39
Energy Saving (×)			
TinyViT-1	**TinyViT-2**	**MobileViT-1**	**MobileViT-2**
1.00	1.00	1.00	1.00
1.88	1.93	1.75	1.80
1.51	1.47	1.38	1.32

Edge Inference: Table 5.6 shows the inference speedup and energy savings for various Vision Transformer models using batch size. The first row is the baseline (1.00×) and the following next rows are the different quantization configurations: 8/8/4, 4/8/4, and 8/4/4, which are the precision levels for weights, activations, and attention, respectively. TinyViT-1 shows speedup of up to 1.92× and energy savings of up to 1.88× compared to the baseline. TinyViT-2 trained using VIET shows higher speedups and energy efficiencies, reaching up to 1.96× and 1.93×, respectively. For the MobileViT models the speed up and energy savings are minimal on the tested devices. The key takeaway from the test and evaluation can be summarized as:

- Optimized Balance: The approximation methods like 8-bit multipliers and variational inference effectively balance model accuracy with significant improvements in energy efficiency, showcasing only a minimal drop in accuracy for substantial gains in GFlops and latency.
- Adaptive Techniques for Diverse Hardware: Tested across various architectures, the approximation techniques offer adaptable solutions tailored to match specific hardware capabilities, enhancing both the scalability and practicality of AI models on devices ranging from CPUs to GPUs.
- Practical Implementation for Edge AI: By integrating these techniques into distributed edge AI frameworks, the research advances the deployment of AI models that are energy-efficient and capable of real-world application, setting a foundation for future AI developments in edge computing and cyber-physical systems.

Summary

This scheme explored the model training and on-device deployment of data and memory-intensive AI models used in perception tasks. The focus is on balancing model performance metrics against energy consumption using optimized software approximation techniques, such as 8-bit multipliers and variational inference. Our evaluations show that these approximation methods offer a promising design space exploration for striking a balance between model accuracy and energy efficiency. The different architectures used for training and testing provide valuable insights into choosing an appropriate approximation scheme for a balanced trade-off. While comparing the model performance, we observe an improved GFlops and latency gain of 5x while reducing accuracy with a margin of 3–4% for the transformer models (a detailed overview and merit of approaches is shown in Table 5.6). The fundamentals from these approximation schemes can be incorporated within a distributed edge AI framework to match the device resources against the AI models for on-device training and inference. Adopting such techniques is essential for sustainable and feasible AI deployment as the AI landscape evolves towards edge computing and cyber-physical systems. By focusing on the trade-offs involved in training and inference on edge devices, this research contributes to the ongoing efforts to make AI models more accessible and practical in real-world applications.

5.4. Scheme 4: AxC with Mixed Precision Bits for Multimodality

Given the challenges presented by data heterogeneity and the computational intensity of standard transformer models, this work aims to reduce the model size and complexity of the baseline architecture (M). An integrated mechanism is proposed, which uses mixed-precision quantization and variational inference, specifically targeting the efficient deployment of resource-demanding Vision Transformer (ViT) models, as detailed in our published work. [4]

The optimized model, denoted as M^* (also called LiteVit), resulting from the proposed methodology, is intended to deliver an improved balance among algorithmic and system-level performance metrics. To enable such performance improvements, several key challenges are first addressed:

1. For two input modalities, what strategies are used to determine optimal mixed-precision quantization values?

2. How does cross-modal bit assignment influence the overall effectiveness of the model?

3. What approaches are used to maintain balanced metrics, such as energy consumption and accuracy?

4. In what manner are optimizations from variational inference and mixed-precision quantization combined?

The following subsections detail the proposed algorithm and address the aforementioned challenges. The analysis considers "RangeVit" as the model to be optimized for

[4]Work Published as: Katare, Dewant, et al. ‘Variational Inference and Mixed-Precision: Approximating Vision Transformers for the Edge’: Journal of Springer Nature Computing.

multi-modal datasets, incorporating properties present in nuScenes or Waymo benchmarks.

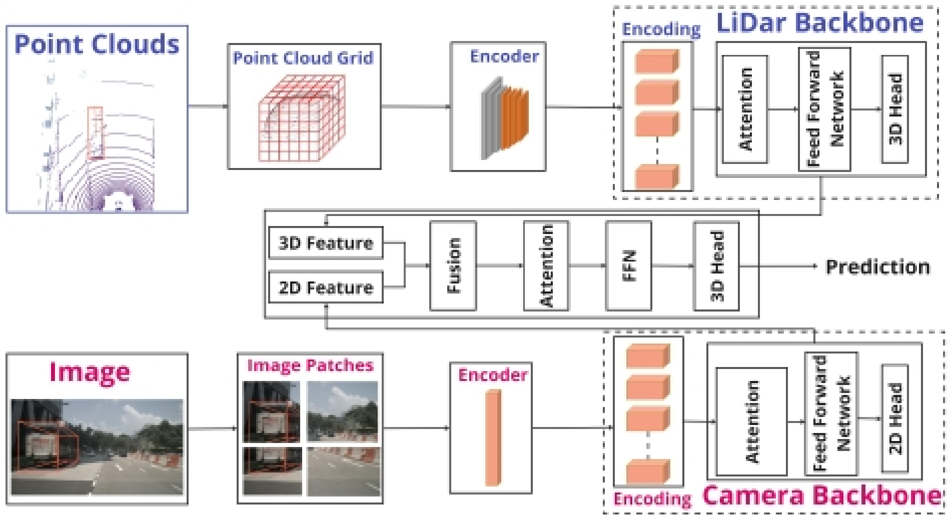


Figure 5.12: Pipeline of multi-modal Vision Transformer.

5.4.1. Overview of Transformer Versions

Transfusion and RangeViT versions mentioned in this chapter, as RangeViT-v1, RangeViT-v2, and RangeViT-v3, are utilized as baselines within the model training and evaluation framework. The distinguishing factor among these versions is primarily the channel size, which directly influences both the number of parameters and the overall model complexity. RangeViT-v1, configured with a channel size of 64, contains approximately 22.7 million parameters, making it the most lightweight of the three. The channel size is increased to 128 in RangeViT-v2, resulting in a parameter count of around 23.7 million. RangeViT-v3 utilizes the largest channel size, 192, and consists of approximately 25.2 million parameters. The purpose of employing these variants is to analyze the effect of model size and complexity on computational efficiency and performance, particularly in environments with restricted resources, such as edge computing. A summary table is included to outline channel size, layers, attention heads, GFLOPs, and total parameter count. Table 5.7 provides an organized comparison of architectural differences across the variants, forming the basis for subsequent optimization and analysis involving mixed-precision quantization and variational inference.

The creation of these models is guided by the hypothesis that changes in model scale can significantly influence the efficiency and performance of Vision Transformers under resource constraints. These model variants will serve as the foundational architectures for further refinement and optimization using the proposed mixed-precision quantization and variational inference techniques.

Table 5.7: RangeVit versions for training via MPQ and Variational Inference.

Model	Chan	Layers	Heads	GFLOPs	Param
v1	64	12	6	21.4	22.7M
v2	128	12	6	25.5	23.7M
v3	192	12	6	27.9	25.2M

5.4.2. Mixed-Precision

Mixed-precision techniques enable different components or blocks within a model to utilize distinct numerical precisions. Higher precision is assigned to critical regions of the model, while less critical parts are processed at lower precision without significantly impacting overall performance. This method serves to decrease memory usage and accelerate computation, which is particularly advantageous for edge devices constrained by memory and processing capabilities. In this framework, the backbone responsible for point cloud processing is configured to operate at lower precision, due to its considerable computational demands resulting from large-scale and complex data. In contrast, the Vision Transformer segment, responsible for 2D image processing, is maintained at higher precision to safeguard the quality of visual feature extraction. By selectively assigning precision levels, resource utilization is optimized [152, 153]. Precision level assignments for various modules and subgroups were guided by empirical analysis and performance evaluation, confirming that this strategy successfully balances computational efficiency and model accuracy.

1) Principles of Mixed Precision: For multi-modal architectures, independent precision tuning for different model components is required. This procedure starts with identifying distinct subgroups associated with each data modality, followed by tuning their respective precision levels independently. The most suitable bit-width for each subgroup is identified by minimizing the loss function relevant to its modality, as formulated in Equation 5.9.

$$\mathrm{BitWidth}(S) = \underset{bw}{\mathrm{argmin}}\,\mathrm{Loss}(\mathrm{Quant}(\mathrm{M}(S), bw)) \tag{5.9}$$

For the above equation, M is the model used for the optimization purpose, S is the subgroup within the model (either camera or LiDAR data that is processed), and bw is the bit-width used for quantization of the subgroup. Loss is the function used to evaluate the effectiveness of quantization mentioned as Quant applied to the weight of subgroup S at a specified bit-width bw.

2) Implementation with Modalities: Within LiTeViT, the backbone dedicated to point cloud processing operates at reduced precision to alleviate computational requirements, whereas the Vision Transformer segment for 2D image analysis maintains higher precision to uphold the quality of visual feature extraction. This subgroup-specific precision allocation is adopted to maximize resource efficiency. Once the optimal bit-width for each subgroup has been identified, quantization is performed as formulated in Equation 5.10.

$$Quantized_S = \text{round}\left(\frac{\text{M}(S)}{\text{scale}_S}\right) \tag{5.10}$$

In this formulation, $Quantized_S$ designates the quantized weights for subgroup S, while $M(S)$ indicates the original weights for subgroup S within the model. Subgroups may correspond to different modules or blocks of the network, such as encoders, where varying precision levels provide an advantage for components that process point cloud data versus those responsible for 2D image analysis. The term scale_S denotes a scaling factor calculated for each subgroup, based on the distribution of weights and the chosen bit-width. This factor standardizes the weight range in subgroup S according to the precision level selected for optimal model behavior. Such scaling ensures that the dynamic range of the original weights is maintained after quantization, preserving essential information even with reduced precision.

3) Cross-Modal Bit-Assignment: The cross-modal bit-assignment mechanism is intended to guarantee that changes to quantization in one data modality (such as LiDAR) support balanced performance metrics, rather than causing degradation in overall model outcomes. This is achieved through an optimization procedure that incorporates joint evaluation metrics for both modalities, as expressed in Equation 5.11, and is further described as follows:

$$BitA = \text{Optimize}(\text{CMP}(\text{Cam}, \text{Li})) \tag{5.11}$$

Here, $BitA$ is the optimized bit-width assignments for subgroups or individual model components. The Optimize function systematically adjusts these bit assignments by assessing their effect through the Cross-modal Performance Metric (CMP). This process evaluates the influence of altered bit assignments during training on the integrated performance and output quality of both the camera and LiDAR processing branches, ensuring that reductions in precision do not negatively impact overall model effectiveness.

4) Loss Function: The composite loss function incorporates multiple terms, each addressing quantization error for camera and LiDAR data separately, and penalizing disparities in performance across the two modalities. This loss formulation guides the model toward an optimal configuration, as defined in Equation 5.12.

$$L = L_{\text{quant}}(\text{Cam}) + L_{\text{quant}}(\text{Li}) + \lambda \cdot L_{\text{reg}} \tag{5.12}$$

Here, L is the total loss function. L_{quant} represents the quantization loss for each modality (Camera or LiDaR). L_{quant} refers to the quantization loss calculated separately for each modality, either camera or LiDAR. The parameter λ serves as a hyperparameter to balance the contributions of quantization loss and the penalty term. L_{reg} is a penalty factor to overcome large differences in performance between the camera and LiDAR modalities. Each component of the loss function contributes to the training process by promoting balanced optimization across modalities as well as regularization. As an example, the values logged at epoch 16 are $L_{\text{quant}}(\text{Cam}) = 0.045$, $L_{\text{quant}}(\text{Li}) = 0.038$, $L_{\text{reg}} = 0.012$, and $\lambda = 0.1$. The total loss L for these values is computed as follows:

$$L = 0.045 + 0.038 + 0.1 \times 0.012 = 0.0842$$

The measured loss values show an early training phase, during which the model begins to reduce quantization errors while maintaining a balance with regularization to minimize overfitting. This stage show the effectiveness of using mixed-precision and quantization strategies in lowering computational complexity without causing a substantial increase in loss or reduction in model accuracy.

Quantization introduces non-differentiable operations, which present challenges for standard gradient-based optimization techniques. To overcome this, gradient approximation methods such as the Straight-Through Estimator (STE) [154] are applied. The STE approach makes it possible to approximate gradients through non-differentiable quantization functions, thus enabling backpropagation and supporting the learning process in a mixed-precision environment. The use of such techniques is essential for efficient optimization of quantized models.

$$\frac{\partial L}{\partial \mathrm{QuantWt}} = \mathrm{STE}\left(\frac{\partial L}{\partial \mathrm{Wt}}\right) \tag{5.13}$$

In the above equation, $\frac{\partial L}{\partial \mathrm{QuantWt}}$ refers to the gradient of the loss function with respect to the quantized weights. The STE (Straight-Through Estimator) method is utilized to approximate gradients through non-differentiable functions. The term $(\frac{\partial L}{\partial \mathrm{Wt}})$ describes the gradient of the loss function with respect to the original, non-quantized weights.

Variational Parameters for Algorithm Approximation

The proposed algorithm combines mixed-precision quantization with variational inference for the training of Vision Transformers (ViTs). This methodology aims to optimize both computational efficiency and model accuracy, with particular emphasis on data-intensive tasks involving large-scale datasets such as nuScenes and Waymo. The training procedure commences by initializing the Vision Transformer model (M) using variational parameters (V), enabling the model to learn a probabilistic distribution over its weights. Adopting a Bayesian approach enhances the capability of the model to address uncertainty and generalize across varied conditions. The following sequence of steps is implemented:

- **Data Processing**: The dataset (D) is partitioned into batches corresponding to camera and LiDAR data, permitting processing strategies tailored to the precision requirements of each modality.
- **Mixed-Precision Quantization**: Individual model components are assigned distinct precision levels, determined by the bit-width range (B), which serves to optimize resource utilization and minimize computational demands while maintaining predictive performance.
- **Loss Computation**: Both training loss (L_{train}) and variational inference loss (L_{var}) are evaluated. The Evidence Lower Bound (ELBO) is incorporated to regularize model complexity. The total loss (L_{total}) is defined as a weighted sum of these terms, achieving a balance between accuracy and complexity.

Here, the fundamentals of the algorithm includes the application of gradient approximation strategies, such as the Straight-Through Estimator (STE), which enable effective backpropagation under mixed-precision settings. Validation is carried out on an independent validation set (D_{val}), and the bit-width assignments for each model component are iteratively refined to achieve optimal performance across both modalities. Upon completion of all training epochs, the final set of model weights (ω^*) is obtained using the optimized bit assignment configuration (O^*). The resulting model is subsequently refined through variational inference, producing an optimized version (M^*) that is well-adapted for deployment in real-time scenarios on edge computing devices with limited computational resources.

Throughout this process, gradient approximation methods such as STE are used to facilitate the training of quantized and mixed-precision models. Performance is continually evaluated on a separate validation dataset (D_{val}), and the bit-width configurations are updated iteratively to further enhance accuracy for both camera and LiDAR modalities. After the training phase, the model parameters (ω^*) are finalized according to the optimal bit allocation strategy (O^*), and further refinement through variational inference ensures that the final model (M^*) maintains high performance and efficiency for edge-based applications requiring real-time inference.

Optimization for Edge: The deployment of LiTeViT models on NVIDIA Jetson Xavier NX requires a sequential optimization workflow, involving both model approximation with size reduction and hardware-specific precision adjustments. Three principal methodologies are implemented to integrate NVIDIA TensorRT with targeted optimization techniques, thereby enhancing the operational efficiency of LiTeViT on the Xavier NX platform.

1) TensorRT Integration and Model Optimization: Initially, the LiTeViT model is converted into a format compatible with TensorRT using NVIDIAs API. This process transforms the high-level architecture into an optimized computational graph, which can be efficiently executed on the Xavier NX hardware. The subsequent precision optimization step utilizes TensorRTs mixed-precision features, adjusting model settings to operate at fp16 and int8 levels. This calibration serves to reduce the memory requirements and accelerate inference speeds. Further optimization is achieved through layer fusion, wherein TensorRT combines multiple computational layers into fewer, more efficient operations. By minimizing inter-layer memory transfers and communication overhead, layer fusion increases computational throughput and lowers inference latency.

2) Advanced Quantization Techniques: Following the conversion, int8 quantization is applied to further compress the model, enabling more effective memory usage and maintaining high throughput on devices with limited resources. Quantization-aware training is performed to ensure that the model remains robust with reduced precision, with model parameters adapted during fine-tuning to accommodate quantization effects. This process is essential for retaining model accuracy after quantization is applied.

3) Dynamic Voltage and Frequency Scaling (DVFS): Dynamic Voltage and Frequency Scaling is employed to adapt the operating frequency and voltage of the Jetson Xavier NX in real time, according to workload demands. This adaptive control manages power consumption and thermal output during inference, aligning DVFS settings with the phases of TensorRT-accelerated execution. Such synchronization guarantees that system resources are allocated efficiently, promoting energy savings during periods of high computational demand. Collectively, these strategies support the deployment of LiTeViT

Algorithm 6 Training Transformers with Mixed-Precision and Variational Inference

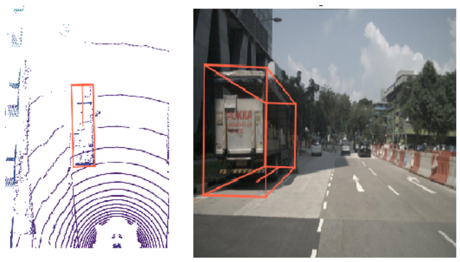
```
Require: Dataset ($D$), Transformer-model ($M$), epochs ($E$), learning rate ($\eta$), bit-width
    range ($B$), variational parameters ($V$), balancing factor ($\lambda$)
Ensure: Optimized model ($M^*$)
 1: Initialize $M$ with variational parameters $V$
 2: for $e = 1$ to $E$ do
 3:     for each $batch$ in $D$ do
 4:         $C_{data}, L_{data} \leftarrow$ Split $batch$ into (Camera, Lidar)
 5:         Process $C_{data}$ and $L_{data}$ through $M$
 6:         Apply Quantization$(C_{data}, L_{data}, M)$ using $B$
 7:         $L_{train} \leftarrow$ Compute Training Loss on $D$
 8:         $L_{var} \leftarrow$ Compute Variational Inference Loss $(M, V)$
 9:         $L_{total} \leftarrow L_{train} + \lambda * L_{var}$
10:         Apply STE for back-propagation
11:         Update weights $\omega$ and variational parameters $V$
12:         Update Bit Assignments for $(C_{data}$ and $L_{data})$
13:     end for
14:     if $e >$ to $E_{val}$ then
15:         Validate $M$ on $D_{val}$
16:     end if
17: end for
18: Derive the final weights $\omega^*$ based on learned optimal bit assignments $O^*$
19: Optimize $M$ based on variational inference criteria
20: $M^* \leftarrow M$
21: return $M^*$
```

models that conform to the computational constraints of Jetson Xavier NX, while ensuring high performance in real-time scenarios.

Experimental Setup

The evaluation focuses on the architecture and configuration of the RangeViT model and its different versions, as detailed in Table 5.7. RangeViT [155] processes both images and 3D point cloud data, which makes it well suited for applications such as high-definition mapping. The model operates on a dual-data stream architecture to manage both camera and LiDAR inputs, incorporating transformer-based self-attention mechanisms to interpret complex spatial relationships. Within RangeViT, a convolutional neural network component projects 3D point cloud data into a two-dimensional representation, reducing computational complexity while preserving crucial spatial features. The integration of 2D and transformed 3D information enhances the model's ability to generate a detailed understanding of the scene.

Dataset and Preprocessing: The nuScenes dataset [112] is widely used for autonomous driving tasks and applications. The dataset includes a diverse collection of urban scenarios captured in both Singapore and Boston, as shown in Figure 5.13a. The dataset contains 1,000 scenes, each approximately 20 seconds in duration, and covers 16 se-


(a) Sample data frame from nuScenes[112]

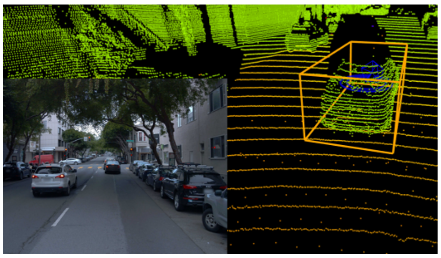

(b) An input frame from Waymo[113]

Figure 5.13: Sample data from the training dataset.

mantic classes through synchronized camera images and LiDAR point clouds. For model training, the preprocessing procedures outlined in the RangeViT methodology are followed, including handling data modalities and partitioning 28,130 training points and 6,019 validation points [155].

Training with Algorithm: The adopted training procedure is tailored for optimizing models for edge computing environments, emphasizing a balance between computational efficiency and model complexity. The Adam optimizer is selected, configured with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a weight decay of 0.0001. A batch size of 32 is used, and the learning rate is initialized at 0.1 and subsequently updated with a cosine decay schedule across training epochs. This approach is intended to achieve robust optimization of the models. After training, each RangeViT variant is converted into its LiteViT equivalent, resulting in substantial improvements in model size, computational performance, and efficiency. These LiteViT models are subsequently assessed with respect to accuracy, latency, and energy efficiency, showing their effectiveness and adaptability for edge computing applications. The transition from RangeViT to LiteViT through this training regimen underscores its suitability for enhancing vision transformers in resource-constrained environments.

Results

The performance of the trained LiTeViT models is compared against established benchmarks on the nuScenes dataset, focusing on comparable metrics such as parameter count, FLOPs, and detection accuracy (IoU).

Model Evaluation: When evaluated on nuScenes, performance metrics such as mean Intersection over Union (mIoU), latency, and GFLOPs reveal that the LiTeViT models, with fewer parameters and reduced computational cost, are able to match or surpass the more complex TransFusion and RangeViT architectures. Table 5.8 provides a detailed breakdown of these results. For example, LiTeViT-1, containing only 10.3 million parameters, achieves an mIoU of 72.9% at a latency of 122.9 ms, showing its suitability for deployment in resource-constrained environments without sacrificing performance.

An additional evaluation of the trained models (all three versions) is conducted on the Waymo validation dataset (see sample in Figure 5.13b), focusing on the object detection task for vehicle (car) and pedestrian classes. Evaluation metrics, summarized

Table 5.8: Evaluation of the trained models on nuScenes validation data, showing FLOPs, Latency, and mIoU.

Model	#Para	GFLOPs	Latency (ms)	mIoU(%)
TransFusion	27.8M	38.3	268.2	69.3
RangeViT-1	22.7M	35.4	188.2	65.1
RangeViT-2	23.7M	38.6	212.5	69.0
RangeViT-3	25.2M	39.4	236.9	74.6
LiTeViT-1	10.3M	24.4	122.9	72.9
LiTeViT-2	16.7M	32.1	145.1	73.8
LiTeViT-3	19.1M	33.7	155.2	74.6

in Table 5.9a, include average precision (AP) and average precision with heading (APH) for both categories. The results indicate that RangeViT models attain strong detection performance, with RangeViT-v3 achieving AP scores of 69.4 for vehicles and 69.9 for pedestrians. The LiteViT variants display a balance between efficiency and accuracy; for instance, LiTeViT-v3 achieves AP scores of 67.4 for vehicles and 72.4 for pedestrians, with APH values of 67.1 and 70.0, respectively. These findings highlight the LiteViT models capability to deliver competitive accuracy while maintaining computational efficiency, supporting their practical deployment in real-world environments.

Energy Evaluation: A comparison of DNN and transformer models, evaluated on the nuScenes validation set, is provided in Table 5.9b. The LiteViT variants, specifically LiTeViT-1 and LiTeViT-3, show an effective balance between accuracy and energy consumption. LiTeViT-1 achieves a mean Average Precision (mAP) of 70.5% while consuming approximately 121.0 mJ, whereas LiTeViT-3 reaches an mAP of 78.3% with an energy consumption of 379.6 mJ. These models are contrasted with the energy-intensive TransFusion, which records an mAP of 66.8% and requires 507.4 mJ to process an equivalent batch size. The RangeViT models show progressive improvements in mAP, with RangeViT-3 attaining 78.3% mAP at an energy cost of 502.9 mJ. Power measurements were obtained using the nvpmodel GUI. The methodology presented for variational inference employs a Bayesian approach during model training, transforming deterministic weights into probabilistic distributions [44, 45]. To further assess the potential for energy savings through variational parameters in the training algorithm, the Laplace distribution is considered as an alternative to the Gaussian distribution in transformers, with its impact on performance and efficiency systematically examined.

5.4.3. Exploring the Laplace Distribution

The Laplace distribution, defined by its mean μ and scale parameter b, is recognized for its heavier tails compared to the Gaussian distribution. The probability density function (PDF) for the Laplace distribution, as applied in this context, is given by:

$$f(x|\mu, b) = \frac{1}{2b}\exp\left(-\frac{|x-\mu|}{b}\right)$$

Within the transformer architecture, model weights are treated as variables sampled

Table 5.9: Model performance results on the Waymo and nuScenes datasets using a batch size of 32.

(a) Results on the Waymo open dataset

Model	Vehicle		Pedestrian	
	AP	APH	AP	APH
Transfusion	58.4	57.9	56.8	54.3
FusionViT	59.5	58.4	54.6	54.8
RangeViT-v1	64.4	63.7	68.9	57.2
RangeViT-v2	68.1	68.2	71.4	66.0
RangeViT-v3	69.4	69.1	69.9	66.8
LiTeViT-v1	61.3	60.6	65.5	64.1
LiTeViT-v2	63.3	62.9	68.8	56.4
LiTeViT-v3	67.4	67.1	72.4	70.0

(b) Energy (mJ) consumption for a batch

Model	mAP	Car	Ped	E(mJ)
TransFusion	66.8	74.1	59.5	507.4
BEVFusion	67.4	71.9	62.9	490.2
VPFNet	62.9	67.1	58.7	495.7
RangeViT-1	74.8	80.3	69.3	416.2
RangeViT-2	75.9	81.7	70.1	480.1
RangeViT-3	78.3	86.2	70.4	502.9
LiTeViT-1	70.5	78.2	62.7	121.0
LiTeViT-2	71.2	76.9	65.5	194.6
LiTeViT-3	78.3	81.8	74.6	379.6

from the Laplace distribution. During optimization, the parameters μ and b are adjusted to improve model fit while controlling complexity, a process governed by the Evidence Lower Bound:

$$\text{ELBO}_{\text{Laplace}} = \mathbb{E}_{q(w|D)}[\log p(D|w)] - \text{KL}(q(w|D)||p(w)),$$

where $q(w|D)$ represents the variational distribution of the parameters w conditioned on the data D, and $p(w)$ denotes the prior distribution.

Comparative Analysis: For the evaluation of variational parameters derived from the variational inference process, the Laplace distribution is considered as an alternative to the Gaussian distribution. Metrics including model accuracy, GFLOPs, and latency are compared in Table 5.10. The findings indicate that the use of a Laplace distribution within the variational inference framework allows LiTeViT models to maintain robust mIoU performance while also achieving reductions in GFLOPs and latency. This out-

Table 5.10: Results using laplace distributions for variational inference.

Model	Distribution	#Params	GFLOPs	Latency (ms)	mIoU (%)
TransFusion	Gaussian	27.8M	38.3	268.2	69.3
LiTeViT-1	Laplace	10.3M	23.0	120.5	72.4
LiTeViT-2	Laplace	16.7M	30.7	142.8	73.5
LiTeViT-3	Laplace	19.1M	31.9	150.0	74.1

come shows an effective strategy for optimizing transformer models for edge computing environments where computational resources are limited.

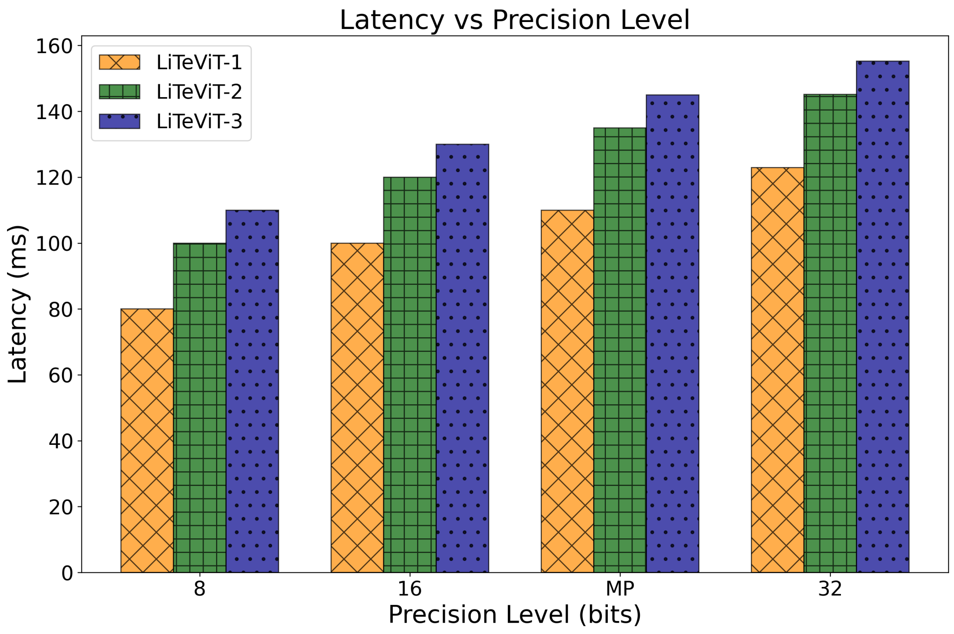


Figure 5.14: Latency Comparison vs Precision Levels.

When latency is used as a function of numerical precision (see Figure 5.14), an inverse correlation is observed. Higher precision configurations lead to increased latency, whereas lower precision settings enhance processing speed. This trend shows the benefits of precision scaling, particularly for real-time applications where rapid data handling is crucial, such as autonomous driving systems. Notably, LiTeViT-1 and LiTeViT-2 achieve reductions in latency at 8-bit precision, supporting the use of reduced precision for non-essential model components. The adoption of a mixed-precision approach is shown to be adaptable to various computational requirements, enabling models to meet performance standards without introducing unnecessary delay. Although these evaluations have been conducted on standard GPU devices, the applicability of these methods can also be further analysed using custom accelerators designed for domain-specific inference tasks.

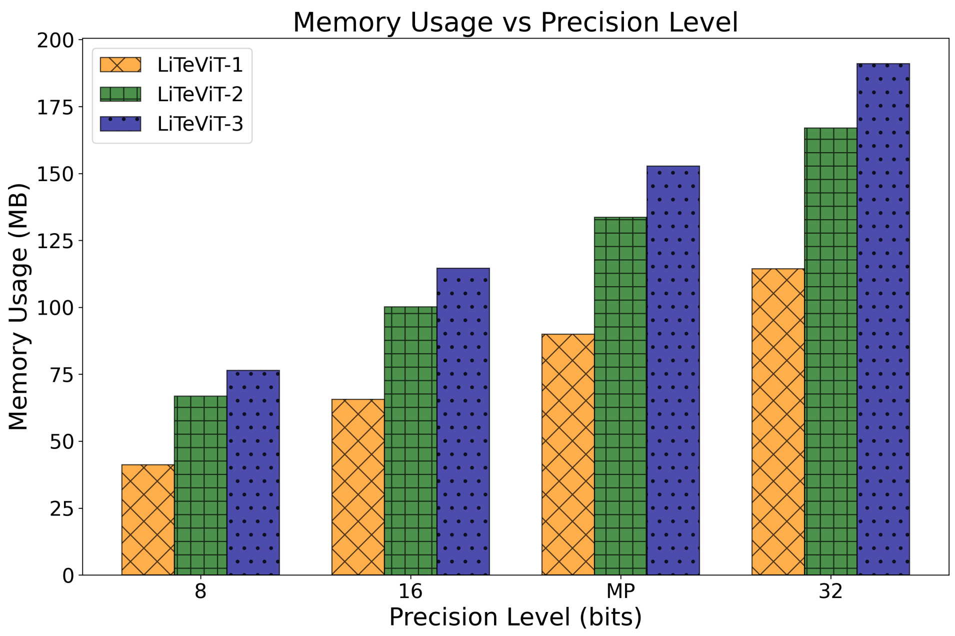


Figure 5.15: Memory vs Precision Levels.

The analysis of memory usage, as depicted in Figure 5.15, highlights how precision adjustments affect system resource allocation. Lowering numerical precision, as implemented in LiTeViT-1 and LiTeViT-3 at the 8-bit level, leads to a reduced memory footprint. This outcome is especially important for edge devices, where onboard memory is often limited. Employing reduced precision enables these models to process larger datasets or support multiple concurrent applications without a major decrease in performance. The mixed-precision technique further enhances this effect by selectively assigning lower precision to non-critical components, thereby optimizing memory consumption while preserving necessary accuracy. This combination contributes to operational efficiency and expands the suitability of these models for real-time and memory-constrained scenarios.

Figure 5.16 presents a comparison of energy consumption across various precision levels for the three LiteViT versions. A reduction in precision from 32 bits to 8 bits corresponds with a marked decline in energy usage, most prominently observed in LiTeViT-1. This trend persists throughout all model variants, confirming the benefit of lower precision for energy efficiency. The adoption of mixed precision (MP) achieves a balance by reducing overall energy demand, while ensuring that higher precision is maintained where it most affects model accuracy. Collectively, these findings show that targeted precision adjustments across different model segments can show improvements in energy efficiency, latency, and memory utilization, supporting the practical deployment of deep learning models in edge environments with limited resources.

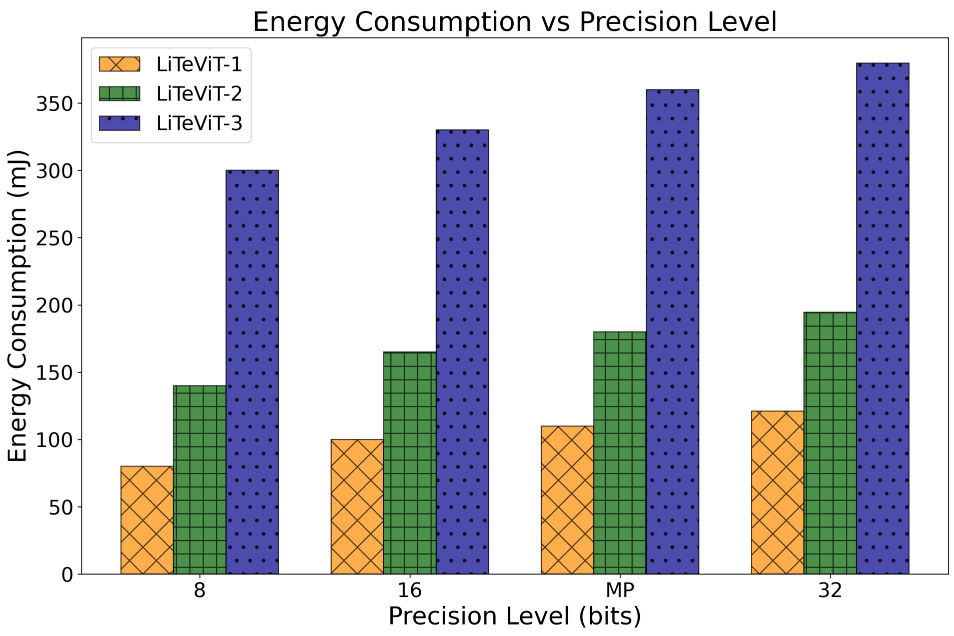


Figure 5.16: Energy comparison for precision levels.

5.4.4. Exploration of Quantization Depth

Extending the methodology developed for training the transformer models with mixed-precision and variational inference, this section investigates the effects of quantization applied at different depths within model layers and modules. The impact on computational efficiency and model accuracy is quantitatively assessed. Quantization is performed by modifying bit-width assignments for high-precision components, such as the feed-forward network (FFN) and multi-head self-attention (MHSA) layers, to alleviate computational requirements without causing a substantial reduction in model performance. Adjusting precision in this manner is essential for achieving efficient deployment on specialized hardware.

A comparison of LiteViT models with mixed-precision quantization settings is further implemented, with configurations utilizing both floating-point and integer representations summarized in Table 5.11. The table presents results for intersection over union, GFLOPs, and latency across various precision configurations, compared with baseline models, to highlight both the improvements achieved and the trade-offs involved.

This analysis shows the efficiency benefits achieved by **LiTeViT-1** using the Mixed Precision A configuration, where a substantial reduction in GFLOPs and latency is observed. These results support the effectiveness of the approach in optimizing computational requirements while maintaining high accuracy. The assessment of the algorithms “Apply Quantization” step highlights its contribution to achieving a favorable balance between computational efficiency and model performance. Future work will focus on automat-

Table 5.11: Performance for quantization strategies (MP = Mixed Precision).

Model Configuration	Precision (Bits)	GFLOPs	Latency	mIoU (%)
Full Precision	32	39.4	268.2 ms	79.7
LiTeViT-1 (MP A)	Encoder: 16, Others: fp8	20.3	100.7 ms	72.4
LiTeViT-2 (MP B)	Encoder: 16, Others: int8	28.9	118.3 ms	73.1
LiTeViT-3 (MP C)	fp 16	29.5	127.5 ms	73.9

ing the selection of precision levels and exploring the integration of mixed-precision quantization with additional model optimization techniques, thereby enhancing the deployment potential of Vision Transformers in environments with limited memory and computational resources.

Table 5.12: Inference speedup and energy saving results (per batch).

Speedup (×)			**Energy Saving (×)**		
LiTeViT-1	**LiTeViT-2**	**LiTeViT-3**	**LiTeViT-1**	**LiTeViT-2**	**LiTeViT-3**
1.00	1.00	1.00	1.00	1.00	1.00
1.84	1.91	1.95	1.72	1.83	1.90
1.29	1.15	1.61	1.21	1.22	1.58

The evaluation of the proposed training approach reveals improvements in inference speed and energy efficiency across all LiTeViT model versions. As presented in Table 5.12, each version exhibits enhanced speed and energy savings compared to the baseline configuration (detailed in Table 5). The baseline is provided in the first row, while the subsequent rows report results for batch sizes of 32 and 64, respectively. Among the three versions, **LiTeViT-3** shows the major speedup and energy savings. The data further show that processing speed and energy savings decrease as batch size increases, suggesting that additional gains could be realized through further optimization of data processing in the model backbone. Collectively, these results confirm the efficacy of the applied optimizations, highlighting these models as strong candidates for use in scenarios where both high performance and limited resources are required.

5.4.5. Summary and Key Takeaways

The training and inference strategies for multi-modal and computationally demanding vision transformer models have been explored in detail. The main conclusions are summarized as follows:

1. **Performance Balance:** The "lite" variants achieve a trade-off between computational efficiency and accuracy. With the proposed hybrid training approach, robust performance is maintained with a reduced parameter count, showing their suitability for use in environments constrained by computational resources.

2. **Energy Efficiency:** On-device evaluations indicate that high accuracy can be obtained while minimizing power usage. **LiTeViT-3** attains the highest accuracy, and

LiTeViT-2 achieves a suitable balance between energy use and accuracy, making it well-suited for applications requiring extended deployment under energy constraints.

3. **Quantization:** Mixed-precision methods have been shown to effectively lower computational and energy demands without a substantial decrease in model performance. In particular, **LiTeViT-2** and **LiTeViT-3** achieved improved speed and energy efficiency compared to the baseline.

5.5. Conclusion

This chapter addressed model-level approximation approaches for vision models used for data-intensive connected vehicular applications. The proposed schemes, including approximate multipliers and variational inference techniques, shows the potential for meaningful reductions in computational workload and energy usage on edge hardware. Integrating floating-point multipliers, probabilistic approximation, and mixed-precision quantization validated the feasibility of sustaining suitable accuracy while reducing the energy and computational overhead. These methods are especially relevant for edge environments, where high performance is needed under limited memory and compute budgets. The chapter also examined the influence of these approaches through practical deployments on selected edge devices. Applications that are non-safety-critical, such as multimedia streaming, can benefit from energy savings by applying these techniques. Achieving a balance between computational precision and energy expenditure relies on efficient resource optimization for sustained application operation. This work contributes foundational insights into approximate computing by systematically evaluating the trade-offs between energy usage, computational throughput, and accuracy, for example, by quantifying how 4-bit, 8-bit, and mixed-precision multipliers influence detector loss, inference latency, and measured GPU/CPU energy consumption on edge devices. Iterative cycles of design and evaluation within the research framework reinforce the practical consequences of these trade-offs in developing energy-aware systems. The outcomes presented here directly support the objectives set out in RQ2 and RQ4, addressing the design and evaluation of essential components. The results in this chapter lay the groundwork for the next phase, particularly chapter 6, which will focus on optimizing trained models for memory efficiency and deploying them on distributed edge devices. Chapter 7 will extend these findings by incorporating both deployment strategies and approximation methods to design a holistic energy-aware framework, ensuring sustainable operation through balanced model performance and energy consumption.

6 M-Alloc: Model Partitions, Resource Allocation and Edge Deployment

Within the edge computing ecosystem, deploying ML models with efficient utilization of computing resources presents an optimization challenge, especially in distributed and heterogeneous computing environments [156, 157]. One of the benefits of using edge AI is the ability to use distributed and heterogeneous computing resources [157, 158]. This benefit complements the domain of connected vehicular ecosystem, where partial computing can be orchestrated on the vehicle and the remaining computing can be carried out on single or distributed edge devices. To fully benefit from this opportunity, designing and developing the model partition mechanisms, model distribution (based on available resources), and adaptive deployment are necessary. While exploring the aforementioned context, this chapter addresses Research Questions 2 (Which components can enable task deployments energy-efficiently and collaboratively in vehicle-edge-cloud computing) and 3 (How can energy-efficient components be integrated into an energy-aware adaptive software framework?) by focusing on designing and developing adaptive mechanisms for model partitioning, resource allocation, and edge deployment.

Through empirical research and exploring serverless edge computing as a use case, this chapter proposes strategies and **technical contributions**[1][2]:

1. **Model partitioning**: The process of dividing ML models into smaller, computationally less demanding sub-models suitable for distributed execution across heterogeneous edge devices. This decomposition aims to optimize for factors such as latency, energy consumption, and communication overhead.

2. **Resource allocation**: The strategic assignment of computational resources (e.g., CPU cores, GPU memory, network bandwidth) to the various components of partitioned ML models to maximize performance and efficiency.

3. **Adaptive deployment**: Dynamically adapting and deploying ML models based on participating device conditions (e.g., workload, memory, and GPU resource availability).

[1]Work Published as: D. Katare, E. Marin, N. Kourtellis, M. Janssen and A. Y. Ding. 'ARASEC: Adaptive Resource Allocation and Model Training for Serverless Edge Computing'. In: IEEE Internet Computing (2024)

[2]Work Published as: Katare, et al. 'Energy-Aware Vision Model Partitioning for Edge AI'. in: Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing. 2025

Recent research has primarily addressed the development of scalable frameworks for model training and inference, with the aim of reducing operational expenses. Although serverless computing and federated learning have been identified as promising solutions, complexities in training optimization for heterogeneous hardware and comprehensive management of costs including computational power, memory usage, resource allocation, and communication overhead are frequently not addressed. In this chapter, the essential elements for distributed machine learning model training and inference are described, with a focus on performance metrics such as accuracy, floating-point operations (FLOPs), parameter count, and latency. The design is tailored for machine learning models deployed in diverse computing environments. The proposed sub-frameworks or components support distributed training processes. For evaluation, the principles of serverless edge computing are applied, and the process of constructing machine learning models from profiled serverless functions, as organized in a look-up table, is examined. Additionally, the estimation of operational expenses for model training in edge, cloud, and hybrid settings is considered. The primary objective is to devise mechanisms for partitioning and deploying models across distributed edge-cloud platforms. Two aims are addressed: first, establishing a connection between model profiling and resource allocation; second, achieving adaptive deployment to enhance resource utilization within heterogeneous edge environments and to develop training strategies that increase efficiency.

6.1. DNN Partitions and Deployment using ARASEC

Deploying machine learning models at the edge requires efficient and optimized usage of resources, including adaptation to a range of hardware (platform or architecture) and network conditions. In distributed and heterogeneous environments, particularly in connected vehicle ecosystems, optimizing model deployment is a multi-faceted challenge. Traditional centralized training approaches can introduce bottlenecks related to communication overhead, energy use, and scalability. Distributed training and dynamic partitioning provide an opportunity to alleviate these issues by allocating computation to the most suitable devices, allowing for lower latency, reduced operational cost, and improved adaptation to changing system conditions. In this context, this chapter introduces ARASEC, a framework designed to partition models, allocate resources, and support adaptive deployment across edge and cloud platforms using serverless technologies. By decomposing models and distributing training workloads, ARASEC aims to improve resource use, manage operational expenses, and maintain performance in terms of both accuracy and latency.

The objectives of model deployment in such a setting can be described as a multi-objective optimization problem, where the training cost (T_c) is a function of resource utilization (U), accuracy (A), latency (L), and model complexity (C):

$$T_c = f(U, A, L, C) \tag{6.1}$$

The objective function can be represented as a weighted sum of their respective factors:

$$T_c = w_U U + w_A(1 - A) + w_L L + w_C C \tag{6.2}$$

where w_U, w_A, w_L, and w_C are weights reflecting the application's priorities for each factor, and can be adjusted based on deployment context.

Systems Overview: The ARASEC sub-framework is built to partition models and enable distributed training across edge and cloud infrastructures, using serverless functions (see Figure 6.1). [3] The workflow begins with the Application component (1) selecting tasks and relevant training data (1.1, 1.2) from the Models Library (2), supporting model formats such as ONNX, MXNet, TensorFlow, and PyTorch. An adaptive partitioning algorithm (3) segments the model using meta, ensemble, or heterogeneous techniques suited to the available hardware and deployment requirements. This step is crucial for managing communication overhead and improving training speed.

During Function Creation (4), model profiling and resource mapping are performed to guide runtime selection and management of serverless functions. The Aggregator validates preparations, ensuring that functions are configured according to Service Level Objectives (SLOs). Post function generation, the model deployment phase (5) manages and optimizes resource assignments. Methods such as Distributed Nesterov Accelerated Gradient (D-NAG) and Asynchronous Parallel Stochastic Gradient Descent (APSGD) are applied to reduce communication costs, operational expenses, and improve convergence. A Command-Line Interface (CLI) enables deployment across platforms in various formats, including containers and web APIs.

The Training Process (6) is managed by a Scheduler, distributing activities across all devices and serverless functions. Aggregation of weights from distributed functions, guided by distributed learning protocols, directly influences model performance. The Metrics profiler (7) tracks accuracy, energy consumption, and other metrics such as precision, recall, and F1-score, enabling optimization of the training pipeline. The Function Zoo (8) catalogs pre-configured serverless functions, supporting deployment across edge and cloud environments. Through this architecture, the framework seeks to balance operational expenses, accuracy, latency, and resource usage, enabling scalable and efficient training for real-world applications.

Processes (Offline, Online): The processes included in the framework can be classified into two main categories. The first, referred to as the offline process, serves as the foundation for generating profiled functions from partitioned models and contributes to the creation of APIs. The second, corresponding to training, represents the standard distributed training procedure, wherein models are trained using serverless functions on edge devices. The modules and components included within these processes are described in detail below.

1. Offline Process: The offline process provides the basis for model partitioning and profiling by concentrating on model and resource metrics. The primary objective is to obtain profiled functions from the partitioned models. As summarized in the overview of the framework, key steps during this phase include Model Partitioning and Profiling, Lookup Table Generation, and Function Zoo Preparation. The offline process results in the development of APIs, enabling automated utilization of the function zoo for model construction and training.

2. Online Process: Following the offline creation of the function zoo, the online pro-

[3]Contributions discussed in published work: Katare et al. 'ARASEC: Adaptive Resource Allocation and Model Training for Serverless Edge Computing'. IEEE Internet Computing (2024)

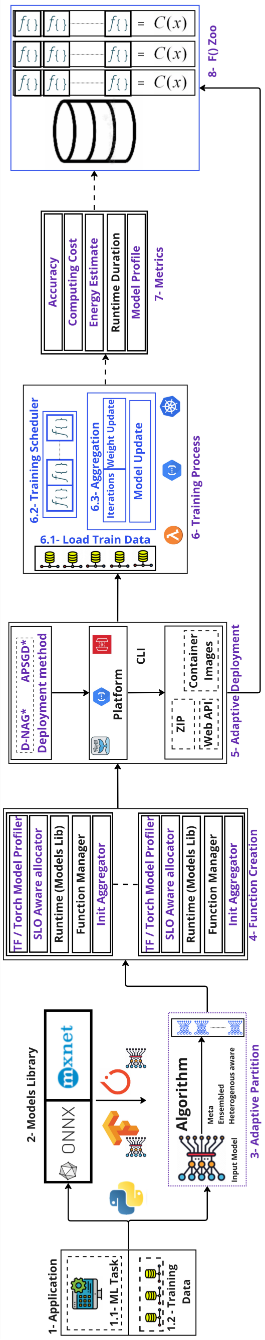


Figure 6.1: Visualization of model partition, resource allocation, and deployment [159]

cess utilizes the API for resource-aware model re-training with new datasets, as well as for model fine-tuning and inference. The online process comprises querying the zoo for functions and resource requirements, retrieving appropriate functions, dynamic resource allocation, model deployment, training, and inference.

6.1.1. ARASEC Components

As shown in Figure 6.1, the ARASEC pipeline consists of multiple modules and components, each responsible for specific functions such as model partitioning, function (zoo) creation, API development, and inference. The key aspects and principles of these components are outlined below.

1. Model Partition: Machine learning models, especially Convolutional Neural Networks (CNNs), are often characterized by high complexity, which presents challenges for execution on individual edge devices. Partitioning these models into smaller submodels enables parallel execution across several edge devices, facilitating efficient training and inference [160, 161]. A common approach involves partitioning each model layer independently (meta partitioning), allowing for layer-wise parallel execution. However, this method may incur considerable communication overhead due to frequent data transfers between layers. In application domains such as vehicular ecosystems, where the advantages of parallelism surpass the associated communication costs, this approach remains practical. Alternatively, the ensembled partitioning technique organizes layers with similar characteristics into blocks, reducing communication overhead by limiting the inter-layer data transfer when compared with the meta approach. This method supports optimized computing and memory utilization by clustering layers that share computational profiles, and it is particularly well-suited for edge-cloud deployments and models containing repetitive layers.

In edge computing scenarios, the heterogeneity of device computational resources often necessitates a partitioning strategy that adapts to available resources. The heterogeneous partitioning method addresses this requirement by dynamically adjusting partitioning decisions based on device-specific constraints and available resources. By taking into account parameters such as CPU/GPU resources (R_{CPU}, R_{GPU}), memory (M), and latency (L), computational resources are allocated efficiently across devices to ensure optimal model execution. The overall partitioning approach is implemented as a two-stage process to prepare models for distributed training. Initially, the Deep Neural Network (DNN) is divided into sub-models by grouping layers with similar computational complexity (C_i). Each resulting sub-model S_i is assigned a weight corresponding to its complexity, which informs subsequent device allocation:

$$S_i = \sum_{j=1}^{n} L_{ij} \cdot C_{ij} \tag{6.3}$$

where L_{ij} represents the layers within sub-model S_i and C_{ij} denotes the computational complexity of layer L_{ij}.

Subsequently, a heuristic optimization, discussed in Algorithm 4, deploys these submodels to devices by balancing the device's memory profile (M_k), computing capabilities (R_k), and inter-device communication latency (L_k). This optimization incorporates the initial complexity weights to maintain model accuracy while ensuring efficient resource utilization:

$$\min \sum_{i=1}^{m} \sum_{k=1}^{d} \left(\alpha \cdot T_{ik} + \beta \cdot L_{ik} + \gamma \cdot M_{ik}\right) \tag{6.4}$$

With the following limits/conditions:

$$\sum_{i=1}^{m} S_i \leq R_k \quad \forall k, \quad M_{ik} \leq M_k \quad \forall i,k \tag{6.5}$$

Here T_{ik} is the execution time of sub-model S_i on device k, L_{ik} indicates the communication latency, and M_{ik} denotes memory usage. The weights α, β, and γ are assigned to balance the significance of execution time, communication latency, and memory usage, respectively. Distributed training is facilitated by first dividing the deep neural network into sub-models containing layers with comparable complexity. Each sub-model is assigned a weight based on its computational complexity to guide suitable device assignment. A heuristic optimization method is then employed to place sub-models onto devices, ensuring that memory requirements, computational capacity, and inter-device communication latency are balanced, while the original complexity weights are preserved to maintain overall model accuracy.

Algorithm 7 Profile Model Components

```
1: function PROFILE_COMPONENTS
2:     profiles ← {}
3:     components ← GET_COMPONENTS()
4:     for component in components do
5:         reqs ← {}
6:         for input_size in input_sizes do
7:             START_PROFILING(component, input_size)
8:             start ← CURRENT_TIME()
9:             output ← EXECUTE(component, input_size)
10:            end ← CURRENT_TIME()
11:            (cpu, mem) ← STOP_PROFILING()
12:            reqs[input_size] ← {'exec_time': end - start, 'cpu': cpu, 'mem': mem}
13:        end for
14:        profiles[component] ← reqs
15:    end for
16:    return profiles
17: end function
```

2. Model Profiling: This phase involves examination of each partitioned model component to determine its resource demands, such as CPU usage, memory footprint, and execution time. Each component is evaluated by executing it with varying input sizes, and the associated resource consumption and timing are measured (as described in Algorithm 7). Here, START_PROFILING and STOP_PROFILING use lightweight system probes (for example, `psutil` or container runtime metrics) to sample CPU usage and peak memory for the profiled component during the execution window between `start` and end. The information collected during this process is essential for effective resource allocation by the SLO Aware Allocator. The result is a detailed report describing the computational features of each component, which is then used to guide later stages of function creation.

3. SLO Aware Allocation: Service Level Objectives (SLOs) define performance metrics that must be achieved by serverless functions, such as response time, throughput, or error rate. Based on the collected profiling data, resources are allocated to serverless functions to ensure sufficient computing power and memory for meeting their respective SLOs, with optimization for both performance and cost (the allocation process is detailed in Algorithm 8). Here, OPTIMIZE_ALLOCATION maps each profile to SLO targets by searching possible CPU, memory, and bandwidth settings and choosing the lowest configuration that meets the specified latency, accuracy, or cost limit for that component. This step requires the determination of suitable memory sizes and execution settings. Additionally, the allocator may combine smaller model components into a single function when such merging results in better resource use, as implemented by the OPTIMIZE_ALLOCATION procedure within the algorithm.

Algorithm 8 SLO Aware Resource Allocation

```
1: function ALLOCATE_RESOURCES(profiles, slo)
2:     allocations ← {}
3:     for comp, profile in profiles.items() do
4:         alloc ← OPTIMIZE_ALLOCATION(profile, slo[comp])
5:         allocations[comp] ← alloc
6:     end for
7:     return allocations
8: end function
```

4. Function Manager: This module is responsible for the lifecycle of serverless functions, including the packaging of model components with all required dependencies and configurations, as well as managing their deployment (as shown in Algorithm 9). Version control is maintained to support updates and enable rollback operations, ensuring system stability and supporting reliable updates.

5. Aggregator: The function of this component is to ensure that all serverless functions are initialized with appropriate parameters and configurations before deployment. Initialization parameters, such as weights, biases, and hyperparameters, are aggregated from the partitioned model components and distributed to the corresponding serverless functions. One of the main challenges involves maintaining state information in an environment that is inherently stateless, which is addressed through methods such as checkpointing or the use of external state management services.

Upon creation and validation, serverless functions are integrated into the serverless architecture. These functions are deployed to respond to designated triggers or events and operate in a distributed fashion to execute computations for the machine learning model. This decentralized framework supports scalable and efficient training of machine learning models across a variety of environments.

6. Optimized Adaptive Deployment: The deployment and orchestration of functions across distributed computing resources, such as edge devices and cloud platforms, are

Algorithm 9 Function Management

```
function MANAGE_FUNCTIONS(components, allocations, states)
    packages ← {}
    for component in components do
        package ← CREATE_PACKAGE(comp.code, comp.dependencies)
        ADD_STATE(package, states[component])
        DEPLOY(package, allocations[component])
        packages[component] ← package
        UPDATE_VERSION_CONTROL(package)
    end for
    return packages
end function
function UPDATE_VERSION_CONTROL(package)
    LOG_DEPLOYMENT(package)
    CHECK_UPDATES(package)
    HANDLE_ROLLBACK(package)
end function
```

optimized using the proposed algorithm 10. Techniques such as D-NAG [162] are utilized for rapid convergence, while APSGD [163] is employed for effective parallel computation, enabling adaptation to dynamic conditions and variable computational capacities.

6.1) D-NAG (Distributed Nesterov Accelerated Gradient): This method refines conventional gradient descent by introducing a momentum term, which accelerates convergence towards optimality through a look-ahead mechanism. The update procedure is mathematically defined as:

$$\psi^{(t+1)} = \gamma\psi^{(t)} + \nabla\mathrm{Cost}(\Phi^{(t)}), \tag{6.6}$$

$$\Phi^{(t+1)} = \Phi^{(t)} - \eta\psi^{(t+1)}, \tag{6.7}$$

where $\psi^{(t)}$ is the velocity, γ is the momentum term, $\nabla\mathrm{Cost}(\Phi^{(t)})$ is the gradient of the cost function, and η is the learning rate.

6.2) APSGD (Asynchronous Parallel Stochastic Gradient Descent): APSGD can facilitate asynchronous updates within parallel computing environments, which meets the requirements of Edge and Cloud resources used for this method. APSGD also improves scalability and fault tolerance. The equation is expressed as:

$$\Phi^{(t+1)}_{\mathrm{global}} \leftarrow \Phi^{(t)}_{\mathrm{local}} - \eta\nabla\mathrm{Cost}(\Phi^{(t)}_{\mathrm{local}})$$

facilitating efficient computation without global synchronization at every iteration.

6.3) Initialization: The procedure commences with the initialization of offloading decisions, establishing the foundation for subsequent iterative refinement guided by performance and cost metrics. For instance, in neural networks, this corresponds to the number of parameters and the execution time on a processing unit.

Algorithm 10 Deployment Using D-NAG and APSGD

```
1: Inputs:
2:   $\lambda$: Load Balancing Factor
3:   $\nu$: Normalized CPU Load
4:   $o_{t,n,i}$: Deployment Ratio for Function$_i$ on Host$_n$ at Time$_t$
5:   $qps_i$: Queries per Second for Function$_i$
6:   $ExecDur_{n,i}$: Execution Duration of Function$_i$ on Host$_n$
7:   Scenarios: Possible deployment options
8:   Paths: DAG Paths for Function Dependencies
9:   LatencyMatrix: Communication Latency between Functions
10: Output: $o_{opt,t,n,i}$: Optimized deployment decisions
11: Initialize:
12:   LearningRate: $\eta$
13:   Momentum: $\gamma$
14:   Parameters: $\Phi$ (e.g., deployment ratios)
15:   Velocity: $\psi$, initialized to zero
16: Procedure:
17: while optimization not converged do
18:     for each Host$_n$ and Function$_i$ in parallel do
19:         Compute gradient $\nabla \text{Cost}(\Phi)$
20:         Update $\psi$: $\psi = \gamma\psi + \nabla \text{Cost}(\Phi)$
21:         Apply D-NAG: $\Phi_{temp} = \Phi - \eta\psi$
22:         Asynchronously evaluate Cost($\Phi_{temp}$) with APSGD
23:         if Cost improved then
24:             Update $\Phi = \Phi_{temp}$
25:         else
26:             Adaptively modify $\eta$ and $\gamma$
27:         end if
28:     end for
29:     Check convergence
30: end while
31: Adjust $\Phi$ to satisfy operational constraints
32: Determine final offloading decisions $o_{opt,t,n,i}$
33: return $o_{opt,t,n,i}$
```

6.4) Iterative Optimization: The primary steps include gradient calculation, where the gradient of the cost function is computed, momentum-based updates utilizing the D-NAG formulation to adjust parameters, and asynchronous updates carried out through APSGD for parameter updates across distributed nodes. After each iteration, operational constraints are verified to ensure the continued validity of the offloading approach. The process is repeated until convergence, resulting in optimized offloading decisions that achieve a balance among cost, latency, and resource utilization. In the algorithm 10, the condition "optimization not converged" refers to a stopping rule based on the cost function. The process stops when either a maximum number of iterations $T_{\max}$ is reached or the relative change in the cost between two consecutive iterations becomes smaller than a small threshold ϵ, that is:

$$\frac{\left|\mathrm{Cost}^{(t)} - \mathrm{Cost}^{(t-1)}\right|}{\mathrm{Cost}^{(t-1)}} < \epsilon.$$

If the cost does not improve for the current update, the step "Adaptively modify η and γ" reduces the learning rate and momentum to stabilise updates, for example:

$$\eta \leftarrow \rho_\eta \eta, \qquad \gamma \leftarrow \max(\gamma_{\min}, \rho_\gamma \gamma),$$

where $\rho_\eta, \rho_\gamma \in (0,1)$ control the decay and $\gamma_{\min}$ avoids a very small momentum value.

7. Look-up Table: By applying the numerical optimization methods outlined above, solutions are obtained for each model partition, leading to efficient resource allocations compiled into a lookup table (for example, see Table 6.1 for ResNet18 and VGG16). This table provides guidance for deploying serverless functions, offering pre-calculated resource configurations and thereby reducing the trial-and-error process associated with resource allocation.

8. Function Zoo: This component serves as a repository of pre-packaged serverless functions, each corresponding to specific machine learning model partitions. It acts as a central collection point for serverless functions derived from partitioned models, supporting rapid access and deployment within serverless computing environments and improving the efficiency of distributed model training and inference. The creation of the Function Zoo consists of several stages:

1. **Function Packaging:** Each model component is encapsulated as an independent, deployable serverless function, with all necessary code, dependencies, and initialization parameters included for execution.

2. **Resource Specification:** Resource requirements for each function are annotated using the lookup table produced during the optimization phase. Specifications include CPU demand, memory allocation, expected latency, and recommended instance type. This guarantees that functions are launched with sufficient resources to meet target performance metrics.

3. **Cataloging:** Functions are indexed in the Function Zoo with unique identifiers and metadata describing their AI tasks, computational attributes, and deployment specifications. This metadata enables efficient retrieval and deployment of functions.

4. **API Integration:** An API is provided for interaction with the Function Zoo, allowing users to query, retrieve, and deploy serverless functions based on particular AI tasks and desired performance outcomes.

9. API: A key outcome of the ARASEC analysis is the API developed for serverless model training, utilizing the Function Zoo described above. The API is responsible for constructing and deploying models in accordance with computing cost and performance metric considerations. Central to the API is the estimation of computational resources, such as CPU, memory, and bandwidth, which directs serverless deployment through resource optimization. Its principal components include the gateway, model constructor, resource optimizer, deployment orchestrator, and moderator.

Table 6.1: Look up table for Approximate Resource Allocation for ResNet18 and VGG16 Model Partitions

Model Partition	CPU Usage	Memory	Latency	Instance Type	Function Spec ID
Conv Layer	15%	512 MB	20 ms	c5.large	Res18-1
Residual Block 1	20%	1 GB	25 ms	c5.large	Res18-2
Residual Block 2	25%	1.5 GB	30 ms	c5.xlarge	Res18-3
Residual Block 3	30%	2 GB	35 ms	c5.xlarge	Res18-4
Residual Block 4	35%	2.5 GB	40 ms	c5.2xlarge	Res18-5
Average Pooling	10%	256 MB	10 ms	c5.large	Res18-6
Fully Connected 1	40%	3 GB	45 ms	p3.2xlarge	Res18-7
Softmax Output	10%	512 MB	15 ms	c5.large	Res18-8
Conv Block 1	20%	512 MB	25 ms	c5.large	VGG1
Conv Block 2	25%	1 GB	30 ms	c5.large	VGG2
Conv Block 3	30%	2 GB	35 ms	c5.xlarge	VGG3
Conv Block 4	35%	2.5 GB	40 ms	c5.xlarge	VGG4
Conv Block 5	40%	3 GB	45 ms	c5.2xlarge	VGG5
Max-Pooling	10%	256 MB	10 ms	c5.large	VGG6
Fully Connected 1	50%	4 GB	50 ms	p3.2xlarge	VGG7
Fully Connected 2	50%	4 GB	50 ms	p3.2xlarge	VGG8
Fully Connected 3	50%	4 GB	50 ms	p3.2xlarge	VGG9
Softmax Output	10%	512 MB	15 ms	c5.large	VGG10

6.1.2. Data Distribution for Training

The implementation of distributed model training is accompanied by the challenge of distributing data across multiple training nodes [164]. Various strategies have been devised for partitioning extensive datasets into smaller subsets to maintain effective learning [165]. During machine learning model training, each node operating a distributed model computes gradients using its local data, and subsequently transmits the computed model weights for aggregation into the final parameters. Optimization techniques are selected according to model structure, data characteristics, and the presence of hybrid or heterogeneous computing resources [166]. In the proposed process, data parallelism approach is implemented as follows:

1. Strategy: The dataset is segmented into smaller portions, each of which can be independently processed by different computing units. This segmentation is performed with consideration for the computational capacities of both edge devices and cloud resources to maximize training efficiency: *Dataset Size* $= N$, *Number of Partitions* $= P$, and *Partition Size* $= \frac{N}{P}$.

2. Model Replication: Replicas of the object detection model are deployed on all participating devices and cloud functions, ensuring that each instance is initialized with identical parameters.

$$\text{Total Models} = M, \quad M = P$$

3. On-device Training: Local training is conducted by each device on its allocated segment of the dataset, involving preprocessing, forward and backward propagation, and gradient computation.

$$\text{Local Update} = \nabla L(\theta)$$

where L represents the loss function and θ denotes the model parameters.

4. Aggregation: Gradients or model updates from all devices are aggregated using a method such as Federated Averaging, facilitating the update of the global model:

$$\theta_{\text{global}}^{\text{new}} = \theta_{\text{global}} + \eta \sum_{i=1}^{P} \frac{N_i}{N} \nabla L_i(\theta)$$

Here η is the learning rate, N_i is the number of samples in partition i, and ∇L_i is the gradient calculated by partition i. The updated global model $\theta_{\text{global}}^{\text{new}}$ is then distributed to all devices, ensuring synchronization and uniformity for subsequent training iterations. Adoption of this parallelism method enables efficient and scalable training of object detection models across distributed environments, utilizing edge and cloud resources to optimize training speed, operational cost, and resource utilization.

6.1.3. API for Functions

The deployment of machine learning models in environments with limited resources, such as edge and serverless platforms, using an API, requires approximate estimation of function resource allocation, runtime duration, and operational cost. A mathematical model is introduced to estimate the resource requirements for various partitions of the machine learning model. As a use case, the VGG16 and ResNet18 models are referenced to show the deployment approach.

1. Profiling: Each partition of the VGG16 model is analyzed to estimate computational complexity (C_i), memory needs (M_i), expected latency (L_i), and CPU utilization (U_{cpu_i}). Profiling is represented by the following function:

$$P(\text{partition}) = (C, M, L, U_{cpu})$$

2. Computational Complexity: The computational complexity (C_i) for each partition is determined by the number of operations, which is dependent on the layer type, input size, and architectural characteristics. Memory requirement (M_i) and latency (L_i) are estimated subsequently based on (C_i) and hardware specifications. The memory needed for a partition is given by:

$$M_i = \text{num\_filters}_i \times (\text{kernel\_size}_i)^2 \times \text{feature\_map\_size}_i \times \text{dtype\_size}$$

3. Resource Usage: CPU utilization (U_{cpu_i}) is estimated based on operation count and hardware attributes:

$$U_{cpu_i} = \frac{\text{ops}_i}{\text{CPU\_speed}}$$

4. Latency: Latency is approximated based on the operation count, memory demand, and available parallel processing:

$$L_i = \frac{\text{ops}_i}{\text{CPU\_speed} \times \text{parallelism\_factor}}$$

5. Optimization: The optimization objective is to minimize the total resource consumption, ensuring that the performance of each partition satisfies predefined criteria. The objective function integrates CPU usage, memory, and latency, subject to constraints on the maximum latency and available memory:

$$\min \sum_i \alpha \cdot U_{cpu_i} + \beta \cdot M_i + \gamma \cdot L_i$$

where $L_i \leq L_{\max}$, $M_i \leq M_{\max}$, and α, β, and γ are trade-off coefficients, with $L_{\max}$ and $M_{\max}$ denoting the latency and memory limits.

6.1.4. Experimental Setup

The computing cost of serverless operations are determined by monitoring resource usage, including function invocations, execution time, memory utilization, and network data transfer. In this approach, edge devices are employed for local data processing, which serves to reduce the need for extensive data transmission. The experimental setup, detailed below, utilizes three device nodes for model training. Resource consumption is logged during training phases, focusing on gradient computation, model updates, and convergence assessments. Metrics such as CPU utilization, memory consumption, and related processing indicators are recorded for each training interval. The evaluation is conducted over 40 epochs, with the average resource consumption captured for each epoch. This information supports a comprehensive comparison of resource demands at various stages of training, identifying the most resource-intensive processes. The methodology for testing and evaluation is outlined below, describing the models, datasets, and environments selected for training and validation.

1) Dataset and Model Selection: To assess the effectiveness of the framework, the MNIST or CIFAR-10 datasets may be chosen for detection tasks, and machine learning models such as ResNet18, SqueezeNet, VGG, and MobileNet are deployed. These models are commonly used for image classification and span a range of complexities, facilitating thorough evaluation of framework performance in diverse contexts. The MNIST dataset is relatively small and tends to yield high accuracy and rapid convergence with the aforementioned large models. Consequently, the CIFAR-10 dataset is used for cloud-based testing and evaluation. CIFAR-10 comprises 60,000 images (32x32 pixels) distributed across 10 classes, making it an appropriate benchmark for measuring the performance of object detection models in computer vision. The dataset's size and complexity provide a suitable test environment for examining the robustness and accuracy of models trained with the proposed approach.

2) Devices: The setup includes three device nodes with distinct specifications. The Raspberry Pi 4 operates with a 32-bit ARM architecture featuring four Cortex-A72 cores at 1.5 GHz, 1 GB RAM, and 64 GB of external storage, and does not contain a dedicated accelerator. The Jetson Nano uses a 64-bit ARM architecture with four Cortex-A57 cores at 1.43 GHz, supplemented by a 128-core Maxwell GPU, 4 GB RAM, and 64 GB of external storage. The Jetson NX is equipped with six Nvidia Carmel cores at 1.9 GHz, a 384-core Volta GPU with 48 tensor cores, 8 GB RAM, and 128 GB of external memory. These devices were selected to represent a range of capabilities for managing various computational loads and storage needs within the test framework.

3) Using Edge Devices: This deployment employs the Distributed Nesterov Accelerated Gradient (D-NAG) algorithm for training models on edge devices. This method is selected for its fast convergence properties. To synchronize model parameters between edge and cloud, Asynchronous Parallel Stochastic Gradient Descent (APSGD) is used, as it is resilient to communication interruptions and delays. This technique is especially effective for dense models that are divided into multiple functions. Serverless functions are deployed on edge devices, enabling model training on locally generated data using lightweight functions. D-NAG is responsible for optimizing the training process, while APSGD supports efficient parameter synchronization. Device heterogeneity and maintaining consistent performance across fluctuating network conditions are recognized as challenges in this deployment. The approach is aligned with the computational requirements of dense models, delivering rapid convergence, minimizing communication overhead, and managing orchestration complexity effectively.

4) Using CloudAWS Lambda: In the cloud scenario, ARASEC utilizes the extensive computational resources and scalability of cloud services, particularly AWS Lambda (an overview is provided in Figure 6.2). Here, model components are allocated to serverless resources in the cloud. D-NAG is employed for global optimization, taking advantage of the comprehensive infrastructure overview available in the cloud. APSGD is used to perform asynchronous model parameter updates, supporting robust and efficient synchronization. This approach is suited for large-scale models and datasets due to its scalability and computational power. The primary challenges involve resource allocation optimization and cost management. Nevertheless, the cloud environment delivers the necessary computational capacity and flexibility for extensive model training and deployment.

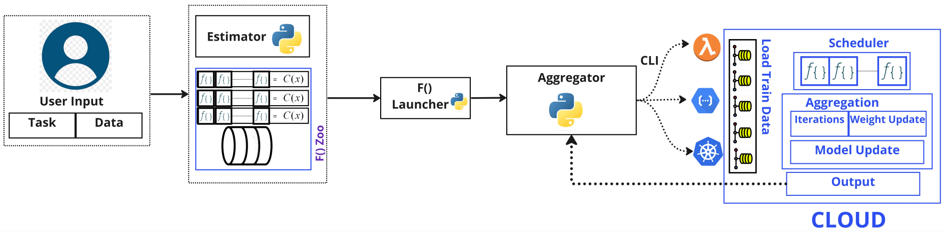


Figure 6.2: Online Training/Inference process (on cloud) for models [159]

5) Edge-Cloud Deployment: This deployment utilizes a hybrid architecture, combining the capabilities of edge devices with those of cloud resources. Through this arrangement, trade-offs between latency, resource usage, and operational cost can be systematically evaluated. Within this configuration, models are trained in the cloud using D-NAG to leverage the substantial computational power available. Following this, APSGD is applied to distribute model parameters to the edge devices, thereby minimizing data transfer at the edge. The hybrid strategy integrates the low-latency, localized processing strengths of edge computing with the scalability and computational capacity of the cloud. Serverless functions are employed to facilitate local computations at the edge and to trigger cloud-based operations for tasks that require significant processing. Cloud resources are activated in response to inputs from edge devices, supporting dynamic and responsive model training. An aggregator component is used to collect and combine updates, maintaining consistent model performance, while a function manager is responsible for overseeing the deployment, scaling, and updating of serverless functions. This approach achieves an effective balance between edge and cloud advantages, fulfilling the requirements of model deployment.

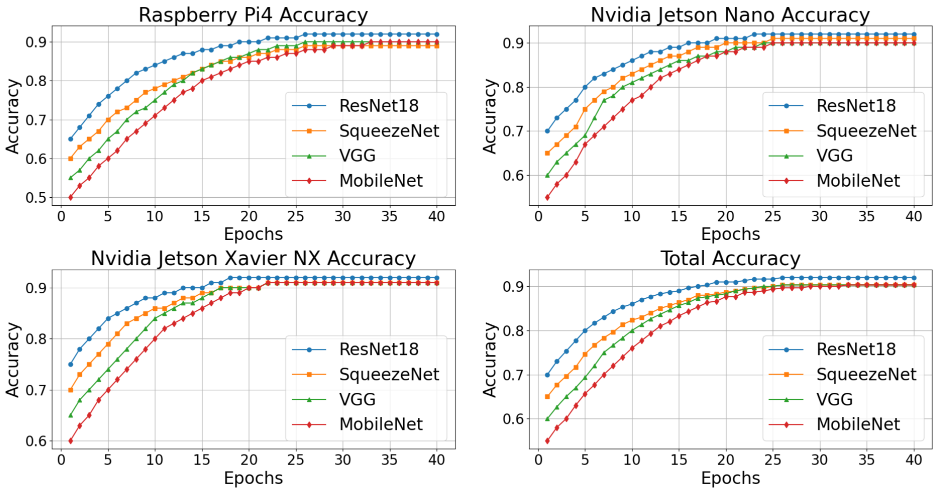


Figure 6.3: Local training accuracy of models on different devices [159].

6.1.5. Experimental Results

The evaluation focused on the effectiveness of serverless edge training, with a detailed analysis of model accuracy, computational latency, and overall cost efficiency. Experiments were conducted using ResNet18, SqueezeNet, VGG, and MobileNet models on the CIFAR-10 dataset, tested across multiple edge platforms. Performance metrics for these deployments are outlined below:

1. Accuracy: As presented in Figures 6.3 and 6.4, the recorded accuracy of each model over training epochs reveals that ResNet18 show superior performance, attaining higher accuracy levels with fewer epochs compared to the other models. For the test configuration used, ResNet18 achieved 85% accuracy within approximately 15 epochs on the Xavier-NX device, whereas VGG and MobileNet required close to 20 epochs to reach a similar performance level. SqueezeNet shows rapid learning initially, achieving 70% accuracy in about 10 epochs across all evaluated platforms, reflecting a relationship between reduced model complexity and accelerated learning. Alternatively, VGG and MobileNet, shows slower convergence, produced more consistent improvements as training progressed. On Raspberry Pi-4, for instance, VGG's accuracy increased from 60% at 10 epochs to 80% after 25 epochs. This gradual improvement may be advantageous for scenarios where incremental learning is required, such as applications involving ongoing data streams, including traffic analysis. The results show that local training on devices such as Raspberry Pi-4, Nvidia Jetson Nano, and Nvidia Jetson Xavier-NX maintains comparable accuracy rates to cloud-based infrastructures, despite resource constraints. This finding highlights the suitability of the model partitioning and distributed training strategy for optimizing accuracy on edge hardware.

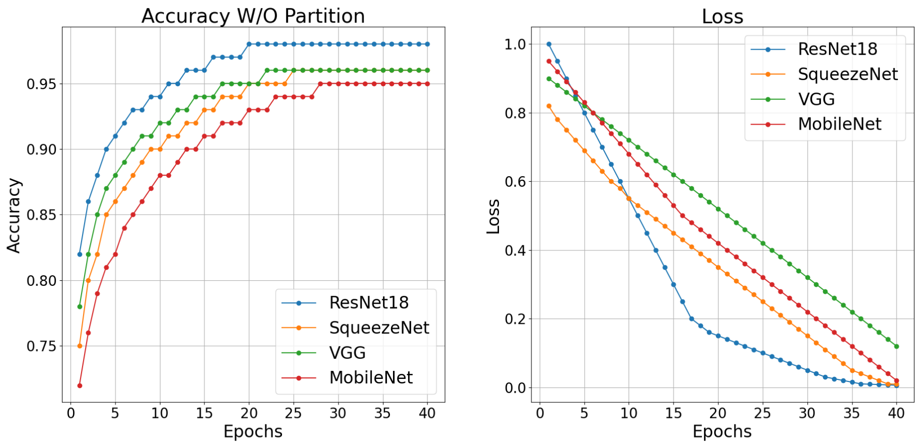


Figure 6.4: Global training accuracy and loss for models [159].

2. Accelerator Impact: Devices equipped with specialized accelerators, such as Nvidia Jetson NX, have been found to deliver improved cost-to-performance ratios. Models with deep architectures, including ResNet18 and VGG16, generally achieve better results on Nvidia platforms, especially Xavier NX, where faster convergence rates are observed when compared with Raspberry Pi. This trend highlights the influence of device computational power on training efficiency. For memory analysis, as depicted in Figure 6.3, the substantial memory requirements associated with models like VGG can be effectively managed on devices offering greater memory capacity, in contrast to the limited resources of the Raspberry Pi 4. Nevertheless, the use of serverless functions enables participation from lower-end hardware, distributing computation more evenly and minimizing the risk of bottlenecks during distributed learning.

3. Edge vs Cloud Cost: The cost evaluation provided in Table 6.2 shows that serverless methodologies can lead to significant reductions in training costs. Serverless deployments yield reduced expenses across various sample sizes and iterations when compared to traditional non-serverless implementations. For smaller batch sizes on less powerful instances, higher costs are observed due to the increased number of iterations needed. This outcome emphasizes the importance of balanced resource management relative to operational expenses. With regard to memory utilization, the estimated cost for AWS Lambda is approximately \$0.205 per hour per batch, which is markedly lower than the calculated \$0.526 per hour. These findings indicate that, when memory costs are appropriately considered, AWS Lambda offers a more cost-efficient alternative to EC2 for these workloads.

4. Limitations: Although the framework supports efficient, resource-aware development and deployment of machine learning models, several limitations remain. The use of a look-up table to record model partitioning, resource allocation, and latency may assume static system conditions, which could reduce adaptability in environments where device specifications and network performance change, potentially causing allocations to become outdated or suboptimal. Additionally, to generalize across various model families, the look-up table must be revised as training requirements shift, since models possess differing characteristics and resource profiles; reliance on a static table may oversimplify this diversity. Scalability is another concern: as the number of models and deployment scenarios increases, look-up table queries may introduce additional latency, which could offset the intended performance advantages. Consequently, periodic updates are needed to accommodate a wide range of models and system metrics.

6.2. LAP-DTR Energy-aware Model Partitions for ViTs

Inspired by the ARASEC framework for CNN, DNN models and respective components, a Layer-adaptive Partitioning with Dynamic Task Redistribution (LAP-DTR) [4] is proposed for partitioning large-scale models such as ViTs [132], DeiT [167], and EdgeViT [168], across edge devices. The approach considers energy consumption and resource allocation for participating devices by ensuring efficient collaborative strategies without excessive communication or model performance degradation.

[4] Work Published as: Katare, et al. Energy-Aware Vision Model Partitioning for Edge AI. in: Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing. 2025

Test Run on AWS Lambda for a limited duration (Lightweight Tasks, e.g., fine-tuning)				
Batch Size	64	128	256	512
Memory (GB)	1.75	1.75	1.75	1.75
Execution Time (sec)	1	2	3	5
Cost (USD / 100ms)	$\$1.67 \times 10^{-5}$	$\$1.67 \times 10^{-5}$	$\$1.67 \times 10^{-5}$	$\$1.67 \times 10^{-5}$
Total Compute Cost (USD)	$\$1.67 \times 10^{-4}$	$\$3.34 \times 10^{-4}$	$\$5.01 \times 10^{-4}$	$\$8.35 \times 10^{-4}$
Test Run on EC2 for ResNet18 model				
Batch Size	64	128	256	512
Instance Type	g4dn.xlarge	g4dn.xlarge	g4dn.xlarge	g4dn.xlarge
Execution Time (min)	60	45	30	20
EC2 Cost (USD / hour)	$0.526	$0.526	$0.526	$0.526
Total Compute Cost (USD)	$0.526	$0.395	$0.263	$0.175

Table 6.2: Cost Evaluation using AWS Lambda and EC2 (Table adapted from [159]

6.2.1. Modeling and Formulation

The LAP-DTR method partitions ViT tasks across heterogeneous devices (CPU, GPU), by dynamically adjusting allocation based on device memory, computational capabilities, and energy consumption (Algorithm 11). This algorithm describes the step-by-step process for the distribution mechanism and reallocation among individual devices, ensuring that each device operates within its optimal parameters. Using Algorithm 11, the method enables real-time inference with minimal accuracy tradeoff, optimizing resource utilization and energy efficiency across the entire system. Detailed problem descriptions and definitions of the equations are covered below.

Energy and Resource Allocation-Aware Partitioning: To ensure efficient deployment, the LAP-DTR framework includes an energy and resource allocation model that guides how the large Vision Transformer is partitioned. For each edge device, the available computational power, memory, and current energy level are factored into the partitioning strategy.

Let E_i be the remaining energy on device i, C_i the computational capacity (in terms of GFLOPs), and M_i is the available memory. The layer partitioning is solved as an optimization problem to minimize overall energy consumption and computational load while meeting latency constraints. The objective is to allocate layers such that:

$$\min \sum_{i=1}^{N} \frac{E_i}{C_i} \cdot L_i \quad \text{such that:} \quad \sum_{i=1}^{N} L_i = L_{total},$$

where L_i is the computational load (number of layers) assigned to device i, and L_{total} is the total number of layers in the Vision Transformer. Each layer l of the transformer has a computational complexity C_l and a memory footprint M_l. The partitioning seeks to ensure that for any device i:

$$C_l \leq C_i \quad \text{and} \quad M_l \leq M_i,$$

ensuring that the assigned layer fits within the device's computational and memory constraints. The framework dynamically updates the partitioning based on the current energy state of each device.

Dynamic Task Redistribution: In addition to energy-aware partitioning, the LAP-DTR framework continuously monitors the workload and battery levels of each edge device during runtime. If a device's energy E_i falls below a threshold E_{min} or if its current workload exceeds its computational limits, the task is dynamically redistributed to other devices.

This redistribution is achieved by solving the following minimization problem, which seeks to balance the computational load across devices while considering their energy constraints:

$$\min \sum_{i=1}^{N} \left(\frac{C_i}{L_i} + \lambda \cdot \frac{1}{E_i} \right),$$

where λ is a weighting factor that adjusts the importance of energy savings relative to computational load balancing. The objective is to offload tasks to devices that have both available energy and computational capacity, avoiding bottlenecks.

Attention-based Feature Summarization: To further reduce communication overhead, an attention-based feature summarization mechanism is used. Instead of transmitting full feature maps, only the most important attention weights and features are shared between devices. Given an attention matrix A generated by a self-attention layer, the feature summarization is performed by applying a compression function $f(\cdot)$ to extract key elements:

$$X_{\text{summarized}} = f(A) \cdot X,$$

where X is the original feature map, and $X_{\text{summarized}}$ is the compressed version that is transmitted between devices. This helps reduce the communication load while preserving the essential information required for accurate inference.

Knowledge Transfer Between Sub-networks: Knowledge transfer across sub-networks is facilitated by passing intermediate representations from one device to another. Each sub-network outputs a feature map X_i on device i, which is used as input to the next sub-network on device j. The transfer function $T(\cdot)$ ensures that the knowledge from X_i is optimally adapted to the sub-network on device j:

$$X_j = T(X_i) \quad \text{where} \quad T(X_i) = W \cdot X_i,$$

where W is a learned transformation matrix that adjusts the intermediate feature map to match the dimensions and structure expected by the sub-network on device j.

Layer Elasticity: To enhance resource efficiency, non-contributing layers of the transformer are rendered elastic. This enables them to dynamically adjust their complexity by reducing the number of attention heads h or the hidden dimension size d in response to the available device resources. The elastic adjustment is given as:

$$h = h_{max} \cdot \left(1 - \frac{E_i}{E_{max}}\right) \quad \text{and} \quad d = d_{max} \cdot \left(1 - \frac{E_i}{E_{max}}\right),$$

where h_{max} and d_{max} are the maximum values for the number of heads and hidden dimension, and E_i is the current energy of device i. As the energy level decreases, the number of heads and dimension size are scaled down, reducing the computational load on the device.

6.2.2. Model Partitioning Profile: ViTs

Vision Transformers (ViTs) are structured differently from traditional CNNs. They consist of layers of multi-head self-attention mechanisms and feed-forward networks, which are known to be computationally demanding. In particular, the attention mechanism requires each input token to interact with every other token, making the computation complexity grow quadratically with input size. For partitioning ViTs, the main objective is to distribute the attention heads and feed-forward layers across the available devices, balancing the workload and ensuring that the computational or memory demands are within device range. Each layer in a ViT can be characterized by the following factors:

- Attention Heads: The number of attention heads in a self-attention layer directly influences the computation cost. Devices with higher computational capacity are assigned layers with more heads.

Algorithm 11 Selecting a Model Partition Strategy

```
Require: Device capability set $C$, required accuracy $Acc$, required energy $E$, required
    latency $L$, weights for accuracy $w_{Acc}$, energy $w_E$, and latency $w_L$
 1: Initialize the list of candidate profiles $Candidates = \emptyset$
 2: for each model partition profile do
 3:     Get the accuracy $Acc_{\text{profile}}$ from the profile
 4:     Get the energy consumption $E_{\text{profile}}$ from the profile
 5:     Get the latency $L_{\text{profile}}$ from the profile
 6:     if $Acc_{\text{profile}} \geq Acc$ and $E_{\text{profile}} \leq E$ and $L_{\text{profile}} \leq L$ then
 7:         Add the profile to $Candidates$
 8:     end if
 9: end for
10: if $Candidates$ is empty then return $profile$ where $Acc_{\text{profile}}$ is $max$
11: end if
12: Initialize the best model partition $BestProfile = \text{None}$
13: Initialize the best score $BestScore = -\infty$
14: for each candidate profile do
15:     Get the accuracy $Acc_{\text{candidate}}$ from the candidate profile
16:     Get the energy consumption $E_{\text{candidate}}$ from the candidate profile
17:     Get the latency $L_{\text{candidate}}$ from the candidate profile
18:     Calculate the accuracy score $Score_{\text{acc}} = w_{Acc} \cdot Acc_{\text{candidate}}$
19:     Calculate the energy score $Score_{\text{energy}} = w_E \cdot \frac{E_{\text{candidate}}}{E_{max}}$
20:     Calculate the latency score $Score_{\text{latency}} = w_L \cdot \frac{L_{\text{candidate}}}{L_{max}}$
21:     Calculate the overall score $Score = Score_{\text{acc}} + Score_{\text{energy}} + Score_{\text{latency}}$
22:     if $Score > BestScore$ then
23:         Update the model partition $BestProfile = \text{candidate}$
24:         Update the score $BestScore = Score$
25:     end if
26: end for
27: Return The model partition $BestProfile$
```

- Embedding Dimension: The size of the embedding dimension affects the memory and processing power required. Larger embedding dimensions increase the complexity of matrix operations, so they are allocated to devices with sufficient resources.

- Feed-Forward Network Size: Each transformer layer includes a feed-forward network that processes the output of the attention heads. The size of this network is a key factor in determining the memory and computation needs.

- Token Count: The number of tokens passed through the layers influences the overall complexity, as the attention mechanism scales quadratically with the number of tokens.

The partitioning profile for ViTs is optimized to minimize energy consumption and maximize efficiency. Layers with more attention heads and larger embedding dimensions are assigned to devices with higher capabilities, while the layers with reduced complexity are allocated to less powerful devices. During runtime, the system dynamically adjusts the partitioning based on the energy availability of each device. If a device becomes low on energy, the framework can reduce the number of attention heads or compress the feed-forward network to allow the model to continue processing without interruption. The partitioning profile for ViTs considers both the computational and memory requirements, ensuring that the workload is appropriately distributed across the edge-device network.

1) Energy-aware Edge Collaboration: As the framework operates in a distributed manner, where edge devices collaboratively perform inference tasks. Intermediate results are cached locally to ensure robustness, and the system adapts to network conditions by redistributing tasks when necessary. Communication is minimized through the attention-based summarization technique, while fault tolerance is ensured by local result caching.

2) Orchestration of Training and Inference: The LAP-DTR framework orchestrates both training and inference phases across multiple devices by dynamically partitioning the model and adapting to device constraints. The following steps show how the orchestration is managed using multiple mixed metrics and the specific steps involved in managing the orchestration.

Evaluation Metrics: The LAP-DTR framework is evaluated using the following metrics for training and inference:

- **Latency**: The total time taken for inference across all edge devices.
- **Energy Consumption**: The energy used by each device during inference, including both computation and communication.
- **Accuracy**: The model's performance in terms of classification accuracy.
- **Resource Utilization**: The proportion of available computational and memory resources used by each device.
- **Communication Overhead**: The data transmitted between edge devices during inference.

Training Procedure: The training process is managed as described in the Algorithm 12, which details the steps for distributing and managing model layers across multiple edge devices. This algorithm ensures that each device is assigned layers based on its available energy, computational capacity, and memory. By following Algorithm 12, the system achieves a balanced load distribution, optimizing energy consumption while maintaining efficient training performance across the entire network.

- The system first profiles each device's available energy, compute capacity, and memory.
- Layers are assigned based on the energy and computational capacities of each device, ensuring balanced load distribution.

Algorithm 12 Distributed Training for Vision Transformers

```
Require: Vision Transformer model $M$, dataset $D$, edge devices $E = \{E_1, E_2, \dots, E_N\}$ with properties energy $E_i$, computation capacity $C_i$, memory $M_i$, and communication bandwidth $B_{ij}$ between devices $E_i$ and $E_j$.
Ensure: Optimized and partitioned model $M$ distributed across $E$.
 1: Initialization: Profile energy, compute resources, and memory for each device in $E$.
 2: for all $l \in \text{layers}(M)$ do
 3:     Compute $C_L$ and $M_L$ for layer $L$
 4:     Assign layer $l$ to device $E_k$ where $k = \arg\min_i \left(\frac{E_i}{C_i} \cdot C_l, M_l \leq M_i\right)$
 5: end for
 6: for all $E_I \in E$ do
 7:     Train sub-model $M_i$ using data $D_i$
 8:     Transfer intermediate feature maps $X_i$ to $E_{i+1}$, where $X_{\text{sum}} = \text{Attention}(X_i)$
 9:     if $E_I$ runs out of energy then
10:         Reassign layers of $M_i$ to minimize energy consumption:
11:         $L_j = \arg\min_j \left(\sum_{j=1}^{N} \left(\frac{C_j}{L_j} + \lambda \cdot \frac{1}{E_j}\right)\right)$
12:     end if
13: end for
```

- During training, intermediate feature maps are transferred between devices using attention-based summarization, minimizing the communication overhead.
- If a device runs out of energy, its layers are redistributed to other devices in the network based on their remaining capacity.

Inference Steps: The inference steps are managed by Algorithm 13, which outlines the procedures for monitoring device states and adjusting layer complexities in real-time. This algorithm enables the dynamic redistribution of tasks and ensures that each device operates within its current energy constraints. By implementing Algorithm 13, the framework maintains robust inference performance and minimizes communication overhead, even in environments with heterogeneous and fluctuating device capabilities.

- For each inference task, the system monitors the workload using Flops, latency and energy levels of each device.
- As an optimized strategy during partition, the algorithm adjusts assigned layers' complexity based on its current energy level (e.g., reducing the number of attention heads or hidden dimensions) for devices.
- Intermediate feature maps are transferred between devices using summarized attention scores to minimize inter-layer data transmission.
- If a device fails or disconnects, the cached intermediate results are used to resume inference from the last state.
- Once all devices complete their assigned tasks, the final result is aggregated using a simple aggregation function G.

The orchestration handles device heterogeneity, ensuring transitions between energy states and task redistribution. The system is highly adaptable and fault-tolerant, providing robust performance even under varying network and hardware conditions.

6.2.3. Complexity Analysis

The computational complexity of this algorithm consists of partitioning and the distributed training/inference phases.

Partitioning Phase: For the partitioning of L layers among N edge devices, each layer's computational complexity C_l and memory M_l must be calculated. This involves $O(L)$ complexity for profiling the model. Assigning layers to the edge devices involves solving a resource allocation problem that depends on the device's energy E_i, computational capacity C_i, and memory M_i. This allocation is done in $O(L \cdot N)$ time, as each layer is assigned based on minimizing the energy-to-computation ratio.

Training and Inference Phase: During training, each device trains its assigned sub-model. The overall time complexity is governed by the communication overhead and the computational complexity of each sub-model. The communication complexity between devices is reduced by summarizing attention features, resulting in $O(N)$ communication rounds, where N is the number of edge devices. Since the layers are partitioned adaptively, the computational load on each device is balanced, ensuring no single device becomes a bottleneck. The overall training time can be approximated as:

$$T_{\text{train}} = \sum_{i=1}^{N} T(M_i) + O(N),$$

where $T(M_i)$ is the training time of sub-model M_i on device E_i.

Inference follows a similar complexity profile, where each device processes its assigned layers, adjusting complexity based on energy availability. The dynamic adjustment of layer complexity results in a time complexity for inference as:

$$T_{\text{infer}} = O\left(\sum_{i=1}^{N} C(M_i) \cdot \frac{E_i}{E_{\max}}\right) + O(N),$$

where $C(M_i)$ is the computation cost of sub-model M_i on device E_i and $O(N)$ accounts for communication overhead.

6.2.4. Test Setup and Multi-Metric Evaluation

We tested the following hardware with the model and dataset setups to validate the proposed energy-aware partitioning approach.

Hardware Setup: We utilized a heterogeneous set of edge devices, mainly CPU and GPU resources, including Nvidia Jetson NX, Jetson TX2, and Raspberry Pi 4. These devices provide varying levels of computational power and memory resources, enabling us to test the adaptability of our partitioning strategies in real-world edge AI scenarios. Each device's energy consumption and latency were recorded during training and inference.

ViT Models: At present, the ViT models can be categorized based on attention mechanisms, FFN (feed-forward network) layers, and the usage of CNN (convolution neural

Algorithm 13 Distributed Inference for Vision Transformers

```
Require: Trained Vision Transformer models $\{M_1, M_2, \ldots, M_N\}$ assigned across devices $E$.
Ensure: Aggregated inference results $Y$.
 1: for all inference tasks do
 2:     for all $E_i \in E$ do
 3:         Monitor current energy $E_i$ and workload
 4:         Adjust layer settings for current energy state:
 5:         $h_i = h_{\max}\left(1 - \frac{E_i}{E_{\max}}\right), \quad d_i = d_{\max}\left(1 - \frac{E_i}{E_{\max}}\right)$
 6:         Execute inference on $M_i$
 7:         Transfer summarized features $X_{\text{sum}}$ to $E_{i+1}$
 8:         if $E_i$ fails or disconnects then
 9:             Resume inference using cached results at $E_{i+1}$
10:         end if
11:     end for
12:     Aggregate outputs from all models: $Y = G(M_1(X), M_2(X), \ldots, M_N(X))$
13: end for
```

network), GNN (Graphs neural network) and MLP (multi-layer perceptron) operations. We evaluated the following Vision Transformer models:

ViT [132]: The original Vision Transformer model that applies transformer architectures to image recognition tasks, leveraging self-attention mechanisms for high performance in various computer vision applications.

EdgeVit [169]: A Vision Transformer variant optimized for edge devices, enhancing computational efficiency and reducing memory usage to enable real-time image processing in resource-constrained environments.

TinyViT [170]: A compact transformer model designed for resource-constrained environments, offering reduced size and computational requirements without significant loss in accuracy.

DeiT-B [167]: A Data-efficient Image Transformer that enhances training efficiency and performance by utilizing knowledge distillation techniques, making it suitable for scenarios with limited labeled data.

EfficientViT [171]: A lightweight version of ViT optimized for efficient inference on edge devices, balancing performance and energy consumption for deployment in low-resource settings.

Tested Datasets: These above ViT models are tested on the OPV2V dataset [168], which is the first large-scale open dataset focused on vehicle-to-vehicle (V2V) communication for perception tasks [168, 172]. This dataset, collected using the OpenCDA framework and CARLA simulator [173, 174], contains: 73 diverse driving scenes across 9 cities and 6 road types. 12K frames of LiDAR point clouds and RGB camera images. 230K annotated 3D bounding boxes. Benchmarks with 4 LiDAR detectors and 4 different fusion strategies (16 models in total). This dataset provides a comprehensive evaluation platform for multi-vehicle sensing and cooperation, which is particularly relevant for autonomous driving scenarios.

Table 6.3: Comparison between Centralized and LAP-DTR using a validation set.

Models	Methods	mAP (%)	Latency (ms)	Energy (mJ)
EdgeViT	Centralized	56.7	4.18	362.5
	LAP-DTR	54.2	3.95	301.1
TinyViT	Centralized	55.3	3.82	418.3
	LAP-DTR	52.7	3.15	376.5
EfficientViT	Centralized	54.8	3.75	450.7
	LAP-DTR	53.1	3.22	392.4
ViT	Centralized	48.3	4.60	481.2
	LAP-DTR	43.8	3.11	413.4
DeiT-B	Centralized	45.9	4.93	421.5
	LAP-DTR	41.4	3.64	351.4

Metrics and Evaluation: We measure three key metrics: Energy Consumption: The total energy usage by each device during inference, measured using external power monitoring tools. Accuracy: The model's performance on 3D object detection tasks using the annotated bounding boxes. Latency: The time taken for model inference across the distributed edge devices.

Figure 6.5 and 6.6, compare the latency and throughput performances of models for a batch across three different devices: Raspberry Pi, Jetson NX, and Jetson TX2. These models include EdgeViT, TinyViT, EfficientViT, ViT, Swin-L, and DeiT-B. In Figure 6.5, latency is measured in milliseconds and shows the time each device takes to process a given input for each model. The Raspberry Pi generally shows higher latency values for all models, suggesting its limited processing power. Figure 6.6 shows throughput, measured as images processed per second, and how each device performs under the workload of different models. The Jetson NX performs better and achieves the highest throughput, which shows its efficiency in managing intensive computations.

Energy Measurements using Tegrastats: To accurately monitor and optimize energy consumption, we utilize NVIDIA Management Library (NVML) for NVIDIA GPUs and Tegrastats for NVIDIA Tegra systems. NVML provides an API for collecting and managing various states of the NVIDIA GPU devices, including power usage, temperature, and utilization rates. This enables detailed tracking and optimization of power consumption during intensive computational tasks. Tegrastats is designed for NVIDIA's Tegra processors, which is commonly used in embedded systems and mobile devices. It provides real-time system monitoring of CPU, GPU, and memory usage and power draw. Using Tegrastats, layer and module level data of energy consumption can be measured which further helps to optimize models for energy efficiency. Figure 6.7 shows energy usage on a device from the point of memory (data flow), CPU, and GPU.

LAP-DTR Sub-framework Summary: LAP-DTR dynamically partitions ViT models to optimize energy efficiency and resource utilization while maintaining acceptable accuracy levels by considering device-specific constraints such as energy availability, computational capacity, and memory. The framework uses adaptive algorithms during training and inference, adjusting model complexity based on real-time device resources available for computing. The approach also addresses device heterogeneity, compute and

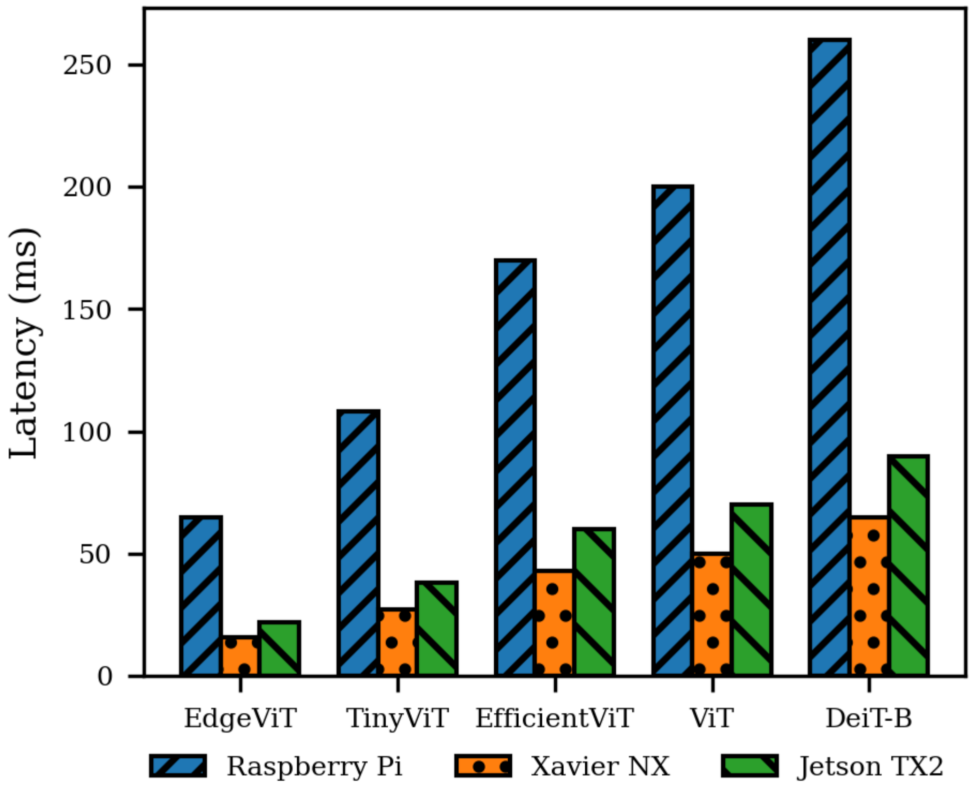


Figure 6.5: Latency comparison for distributed models

memory conditions, providing a robust, error-tolerable solution for distributed ViTs on the edge. The experiments on various ViT models show that LAP-DTR can reduce energy consumption by up to 17% and decrease latency while balancing baseline accuracy. These results shows the practicality of LAP-DTR in real-world edge AI scenarios, especially for applications where energy resources are limited and minor accuracy trade-offs are acceptable. While LAP-DTR mechanism shows potential, few areas require further exploration:

1) Extending to Other Architectures. Using LAP-DTR for different deep learning and transformer models, such as vision language models, to further test adaptability.

2) Model Approximation Strategies. Combine model partitioning with bit-wise approximation techniques to achieve greater energy savings and balance trade-offs for connected vehicle applications.

3) User-Centric Optimization. Incorporate user preferences and quality-of-service requirements to balance energy consumption, latency, and accuracy.

Addressing these areas will enhance the LAP-DTR framework, making it more adaptable and effective for a wider range of applications. By implementing strategies such as model partitioning, API-based integration, and cost-efficient training techniques, the

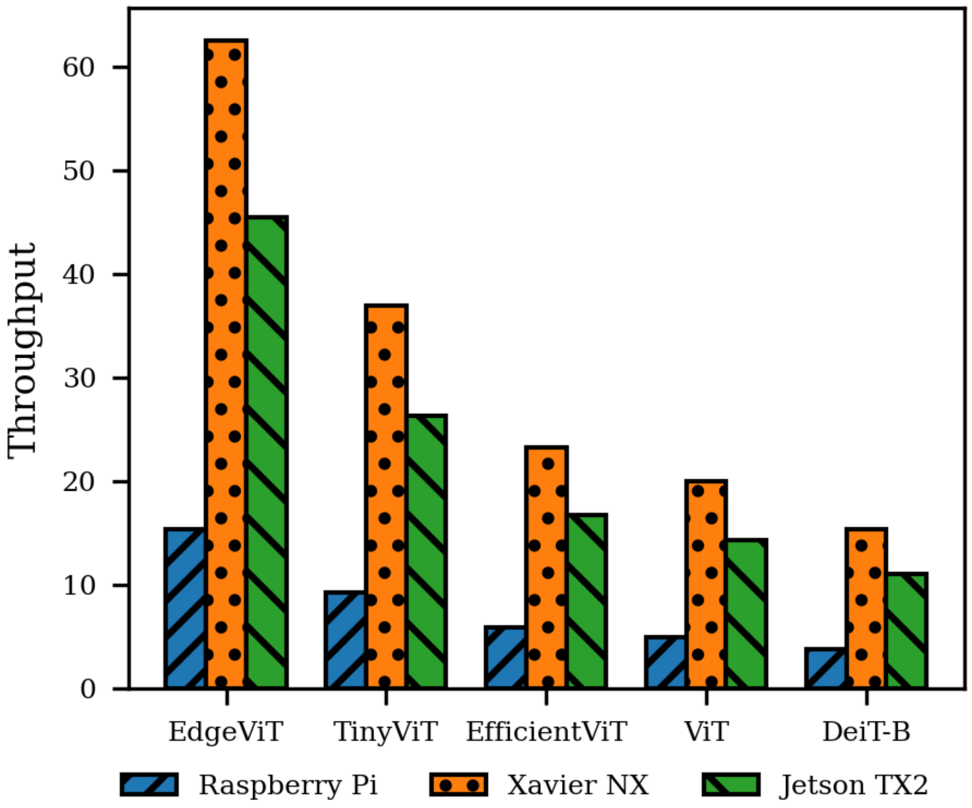


Figure 6.6: Throughput comparison for distributed models

components enable effective deployment of ML models in edge-cloud configurations, leveraging the advantages of serverless functions. Our evaluations show the modules and components' capacity to enhance scalability and reduce costs across various model architectures. The insights from the model partition and deployment mechanism and the energy-efficient evaluation enable us to profile the AI model against the model performance metrics, thus making them suitable as fundamental components ready for integration with the energy-aware adaptive framework.

6.3. Conclusion

This chapter discusses the memory and processing requirements challenges and solutions for efficient training and inference of vision models in distributed computing environments. ARASEC and LAP-DTR sub-frameworks were designed and proposed to partition the models adaptively and optimize their deployment across heterogeneous hardware by profiling models against computing and resource metrics. The outcome of this chapter provides two components, i.e., model partition mechanisms, resource allocation, and adaptive deployment mechanisms. These two components, when combined

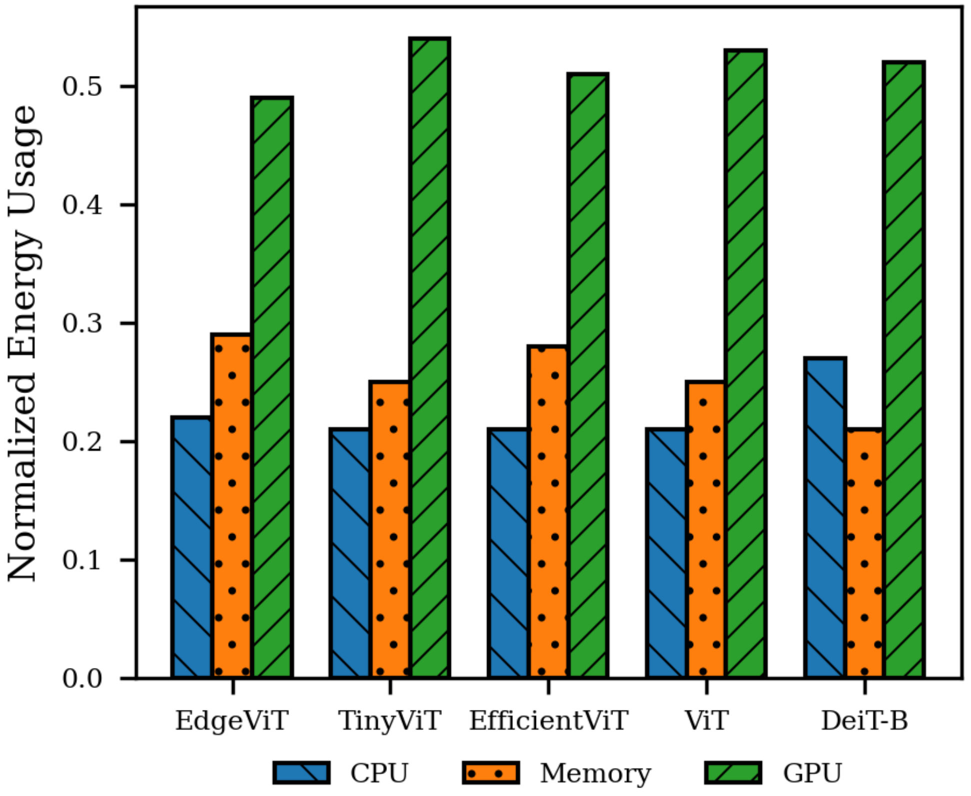


Figure 6.7: Energy Comparison for models on Jetson NX

with previous discussed approximation schemes serve as key contribution and core component for model training and inference tasks in an adaptive energy-aware framework discussed in the next chapter. The experimental tests use object detection tasks with a detailed analysis on resource efficiency, cost-effectiveness, accuracy and latency performance metrics. ARASEC focuses on model partitions and deployment in extreme resource-constrained setting, i.e., serverless edge computing using collaboration of edge devices and cloud. Meanwhile, LAP-DTR is proposed for deploying high-memory and compute demanding models such as Vision Transformers (ViTs) on heterogeneous edge devices. The next chapter will build on these foundational insights to present a comprehensive framework, aiming to provide an energy-aware adaptive software framework for efficient training and inferences using model accuracy, computational efficiency, and energy as key metrics.

7
AxC-Energy: Energy-aware Framework Design

Connected autonomous vehicle services and systems are supported using AI models and computational complex algorithms, which process and analyze heavy data volumes. High-performance computing units and large memory systems support these models, algorithms, and applications, which results in additional onboard energy consumption. The current trend is also towards full electrification of vehicles and increasing connectivity in the vehicular ecosystem to support collaborative and distributed applications using vehicle-edge-cloud computing [175]. However, with the increased focus on model performance and improving the accuracy of these models and applications, the issue of large memory demand, high-performance processing requirements and resulting energy consumption are overlooked. The problem becomes more challenging and complex for resource-constrained edge devices, which are battery-dependent and have limited memory and computing power. This chapter proposes an energy-aware adaptive framework to reduce energy consumption by balancing model accuracy. The discussion of this chapter is based on publication. [1]

The chapter aims to answer research question 3 *(*How can energy-efficient components be integrated into an energy-aware adaptive software framework?) and the research question 4 *(*Can the framework effectively balance the trade-off between energy efficiency and performance in vehicle-edge-cloud computing scenarios?), while discussing the integration of model partition mechanisms, adaptive deployment across edge devices and approximation strategies for the models discussed in the chapter 4, 5, 6 respectively. By integrating these foundational components, this framework uses reinforcement learning as an energy management mechanism for tasks to support energy-aware development across edge devices and platforms. This chapter introduces an adaptive, energy-aware framework for connected autonomous vehicles (CAVs) operating in vehicle-edge-cloud environments. The framework integrates model partitioning, approximation methods, and adaptive deployment to address the energy and computational constraints of edge devices. Using reinforcement learning for dynamic energy management, the system is evaluated with vision models on real-world hardware. Results are analyzed using reduced energy consumption and balanced performance, showing the practical benefits of the proposed approach for future CAV applications.

[1] Work Published as: D. Katare, M. Janssen and A. Y. Ding. 'Energy-Aware Adaptive Framework for CAV', 2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC) (IEEE CCWC 2025)

7.1. Overview of Layers in LOPECS Framework

Lopecs is a multi-layered architecture designed to optimize energy consumption and improve computational efficiency in autonomous vehicles (an overview of Lopecs was shown in Figure 2.3). At the first layer, The framework includes input as sensors or incoming data pipelines, followed by the Quality of Experience (QoE) Oriented Service Classification as the first layer, which prioritizes tasks based on their impact on user experience. This layer ensures that safety-critical tasks receive the required computational resources on a priority basis. The next layer is the Runtime Layer, which includes: *Real-Time Operating System (RTOS)* a foundational OS that supports all lower-level operations with minimal latency, *Heterogeneity Aware Scheduler* which manages task allocation across various computing units, ensuring that each task is processed on the appropriate hardware to maximize energy efficiency, and lastly an *OpenCL API + Runtime*, which allows the use of a standardized API for parallel computing across heterogeneous platforms, enhancing the flexibility and efficiency of the system.

The third layer described in the Lopecs framework is *Heterogeneous Computing Platform*, which includes computing support for multiple *CPU Systems* to handle standard computational tasks, with energy-efficient scheduling managed by the Heterogeneity Aware Scheduler, and *GPU Units*, which include specialized 3D GPUs and an Image Processor, for high-intensity computation tasks such as image processing and complex tasks such as SLAM. This layer also includes support for *Dedicated Accelerators* such as video and audio accelerators, which process specific tasks related to video and audio processing and power usage for multimedia. Lastly, one of the most essential layers in Lopecs is the Vehicle-Cloudlet Coordinator, which manages the communication between the vehicle's onboard computing system and other edge computing resources (cloudlets). This coordinator dynamically offloads tasks to edge servers based on current network conditions and system load, enhancing the overall energy efficiency by using computing resources available within the vehicle-edge ecosystem. Integrating these components together as a framework addresses the autonomous vehicles' onboard power consumption challenge and supports driving services with a distributed, adaptable approach. The evaluation for Lopecs included testing multiple services with a total power consumption of 11W on the Jetson TX1 device. Overall, the framework is the proof-of-concept edge computing system implemented and tested on edge devices with autonomous vehicles, addressing the necessity for real-time, power-efficient computing solutions. While LoPECS included components for reducing power consumption, its performance relies on the proximity of edge cloudlets, which may only be consistently available in some environments. The framework also requires expanding the applicability to a broader range of computing architectures and driving scenarios. Additionally, addressing the problems associated with ML model partitions, data processing mechanisms, and adaptive deployment for inference on vehicle-edge scenarios remains an open challenge.

7.2. Proposed Framework (AxC Energy)

The components of the adaptive software framework discussed in chapters 4, 5, and 6 were designed to deploy in vehicular services-related vision models. Figure 7.1 shows layer aligned components. The contributions in the framework are from the following

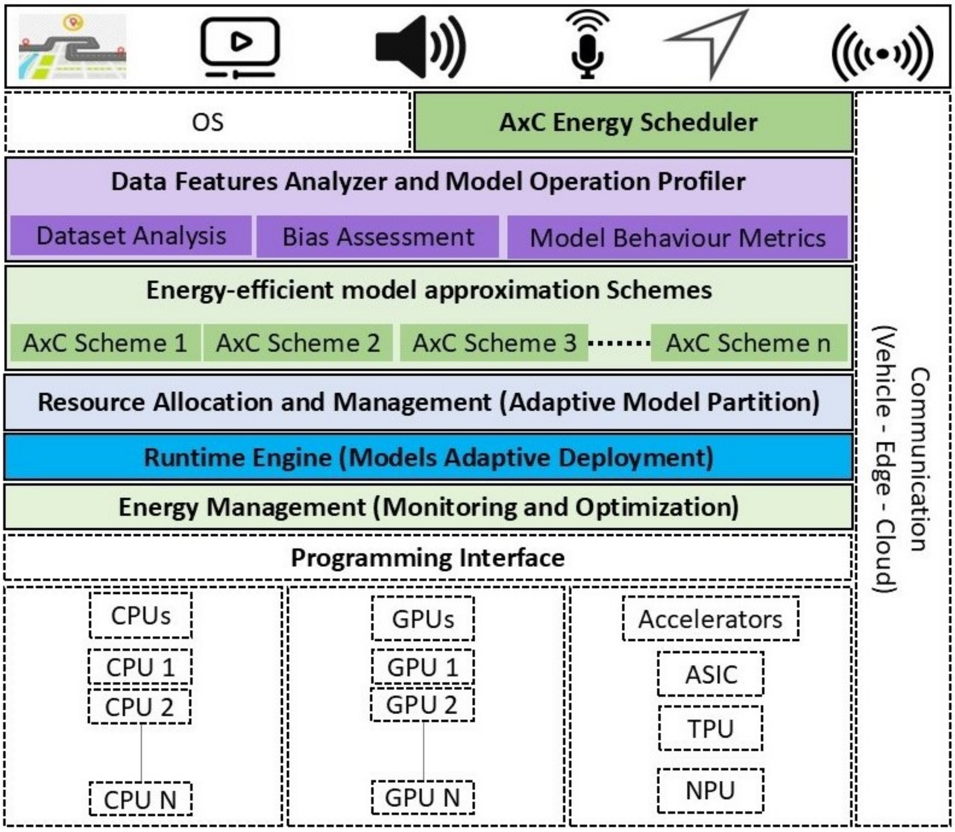

OS
AxC Energy Scheduler
Data Features Analyzer and Model Operation Profiler
Dataset Analysis
Bias Assessment
Model Behaviour Metrics
Energy-efficient model approximation Schemes
AxC Scheme 1
AxC Scheme 2
AxC Scheme 3
AxC Scheme n
Resource Allocation and Management (Adaptive Model Partition)
Runtime Engine (Models Adaptive Deployment)
Energy Management (Monitoring and Optimization)
Programming Interface
CPUs
CPU 1
CPU 2
CPU N
GPUs
GPU 1
GPU 2
GPU N
Accelerators
ASIC
TPU
NPU
Communication (Vehicle - Edge - Cloud)

Figure 7.1: An overview of proposed adaptive energy-aware framework

layers:

1. Services Scheduler: Describes the implementation of a real-time operating system that supports multitasking and resource-aware scheduling. The scheduler is optimized to handle high-priority tasks such as emergency response and navigation with minimal latency, while the runtime engine supports on-the-fly reconfiguring system parameters to adapt to changing environmental and operational conditions. The contribution is discussed in chapter 6.

2. Vehicle Task Profiler (Services Prioritization): This module analyzes incoming data and service requests to prioritize tasks based on latency, resource availability, and energy consumption metrics. This profiler helps in efficient resource allocation by prioritising tasks, ensuring that latency-related tasks are addressed promptly while latency-tolerable tasks are scheduled based on runtime to optimize overall energy usage.

3. Resource Allocation and Management: Implements a dynamic resource manage-

ment system that uses predictive load balancing and real-time monitoring to optimize the distribution of computational tasks across CPUs, GPUs, and dedicated accelerators. This layer uses machine learning algorithms to predict workload patterns and adjust resources using reinforcement learning by reducing idle times and memory overhead. The contribution is discussed in chapter 5 and chapter 6.

4. Approximation Schemes (AxC Schemes): This layer contains optimized algorithms and computing schemes like low-precision arithmetic and probabilistic computing to reduce the system's computational overhead and memory requirements. The approximation schemes are proposed based on the incoming data, model and computing requirements. Detailed implementation of techniques such as variational inference, stochastic rounding, and dynamic precision scaling are discussed in chapters 4 and 5, which discuss their integration into the systems processing units to reduce power consumption.

5. Models Adaptive Deployment: Provides a deployment strategy for ML models within the edge computing ecosystem, which includes vehicles, edge servers, and the cloud. This strategy is implemented through a resource allocation mechanism to effectively manage heterogeneous hardware requirements. Both synchronous and asynchronous deployment methodologies are used to optimize execution across platforms.

6. Energy Management (Monitoring and Optimization): This component balances computational performance and energy usage. It integrates energy optimization for monitoring power consumption across participating devices within the framework. The energy management mechanism uses this data to adjust using reinforcement learning, optimizing energy use while balancing performance metrics. This function supports the sustainability goals of the system and enhances the overall efficiency of the deployed AI models.

The framework's design maintains modularity and scalability, allowing for easy integration of new services and updates, making it future-proof and adaptable to emerging vehicular services and systems while keeping the energy-aware functionality.

7.3. Design and Implementation

The technical specifications and description for each component within the proposed framework are described here, focusing on high-level architectural decisions and low-level implementation details. This includes *Algorithmic Implementation*: For memory allocation and processing resource management, detailed pseudocode is provided for the algorithms (chapter 6) responsible for task allocation based on predictive analytics. These algorithms analyze legacy data and real-time inputs to efficiently allocate computational resources, minimizing energy consumption while balancing performance metrics using tradeoff. *Integration of Approximation Techniques*: Technical specifications on integrating approximation techniques into the computational pipeline are discussed. This includes multiple approximation schemes. Information on each component follows.

7.3.1. Components and Framework Overview

The proposed adaptive framework,[2] builds upon the principles and fundamental components discussed in Lopecs [74]. The general components of an energy-aware framework include optimizing energy efficiency using an onboard task optimizer and vehicle-edge communication coordinator. A high-level overview of the proposed framework is shown in Figure 7.1. The beginning layer consists of sensors or incoming data pipelines, followed by a service scheduler, which operates on a real-time operating system customized for automotive applications, enhancing application management through multitasking and adaptive time slicing, focusing on energy efficiency. The Vehicle Task Profiler follows these layers while assessing and categorizing incoming data and service requests based on latency requirements, resource availability, and energy consumption profiles. This evaluation is necessary for ensuring that latency-critical and safety-critical tasks, such as collision avoidance, localization and mapping, are processed on priority, while less critical tasks, such as over-the-air updates, HD-map updates or infotainment, are scheduled based on respective latency requirements and where current energy usage can be optimized. These layers are followed by the fundamental programming interface and abstraction of computing and processing units, which provide information on available memory and processors.

The next is the resource allocation and management layer, which is responsible for calculating compute requirements for the tasks and their dynamic distribution of computational resources such as CPUs, GPUs, and specialized accelerators like TPUs and NPUs to maximize resource utilization and energy savings. This subsystem uses predictive load balancing and dynamic voltage and frequency scaling (DVFS) to effectively adapt to current workloads and environmental conditions. The next layers are energy-efficient approximation schemes that optimize the models for energy efficiency and accuracy. This layer is followed by adaptive deployment of the model across edge devices; the Energy Management component also centralizes monitoring and optimization efforts, utilizing sensors for real-time energy tracking and machine learning for predictive energy modelling. It adjusts power budgets dynamically and manages task migrations between local and edge computing resources to maintain optimal energy usage. Together, these components form a robust framework that meets the computational demands of CAVs and advances energy-aware computing practices to enhance vehicle efficiency and sustainability.

7.3.2. Model Partition Mechanism

Model partitioning is an important component in our proposed energy-aware framework. This component is designed to distribute large-scale AI models, such as CNN, Vision Transformers (ViTs), across various edge devices efficiently while optimizing energy use and maintaining performance. Our framework introduces the Dynamic Resource-Aware Partitioning (DRAP) strategy combining multiple model partition mechanisms discussed in chapter 6, which adapts edge devices' diverse capabilities for tasks. DRAP considers each device's computational power, memory availability, and energy levels; it

[2]Work Published as: D. Katare, M. Janssen and A. Y. Ding. 'Energy-Aware Adaptive Framework for CAV', 2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC) (IEEE CCWC 2025)

also analyzes the structure of models such as ViTs to identify optimal points for model splitting. Additionally, it evaluates the latency and accuracy requirements of different CAV tasks to guide partitioning decisions. DRAP aims to enhance the distribution of model layers across devices by solving a multi-objective optimization problem. The goals are to minimize overall energy consumption, balance computational loads, and meet latency requirements for timely operations. The optimization function is expressed as:

$$\min \sum_{i=1}^{N} \left(\alpha \frac{E_i}{C_i} + \beta \frac{L_i}{L_{total}} + \gamma T_i \right)$$

where E_i indicates the energy consumption, C_i the computational capacity, L_i the number of layers assigned to device i, L_{total} the total number of layers, and T_i the processing time on device i. Coefficients α, β, and γ are weighting factors that balance the priorities.

DRAP includes techniques to reduce complexity and enhance energy efficiency. Dynamic head pruning, for example, adjusts the ViT number of attention heads in ViT layers according to the device's energy state. Elastic dimension scaling reduces the embedding dimensions of layers when energy is limited to decrease computational demands. Selective token processing is used in resource-constrained scenarios, focusing only on the most critical tokens to minimize computational efforts. Effective communication between devices is essential for energy-aware partitioning. DRAP improves this by applying compression algorithms to feature maps, reducing the volume of data transferred. It also adjusts the precision of transmitted data based on available bandwidth and energy resources. Predictive prefetching uses historical data and current context to predict and prefetch necessary data, reducing latency. DRAP continuously adjusts its strategies based on real-time system conditions. If a device's energy level falls below a certain threshold, tasks are reallocated to devices with more energy. It also dynamically manages workload distribution to avoid overloading any single device. Partitioning strategies are adjusted based on current energy and computing resource conditions, prioritizing critical tasks when needed. These strategies ensure that the framework effectively utilizes resources across heterogeneous edge devices in the vehicular ecosystem, allowing CAVs to use complex AI models efficiently while optimizing energy consumption.

7.3.3. Model Approximation Techniques

Energy-aware framework also integrates multiple model approximation techniques to reduce onboard system energy consumption in connected autonomous vehicles (CAVs) while ensuring the functionality necessary for their operation remains intact. These techniques are discussed in detail in chapter 5, and they target the reduction of computational complexity and power demands without substantial impact on the performance of the models. The approximation technique minimizes multiplication in convolutional operations, including stochastic methods to minimize energy use. This includes implementing probabilistic kernel application, where convolutional kernels are applied with varying probabilities determined by their relevance to the specific task, thereby avoiding uniform application across all scenarios. We dynamically adjust the sampling rate for these operations to adapt to the system's current energy state. Additionally, sparse activation maps are generated by selectively zeroing out activations probabilistically, which helps to maintain essential features while reducing the computation.

For inference efficiency and uncertainty measures, our framework also uses variational inference techniques optimized for energy conservation. Low-rank approximations simplify covariance matrices, reducing both memory demands and computational overhead. Adaptive sampling for Monte Carlo estimation allows for adjusting the number of samples based on the system's energy budget, facilitating a balance between accuracy and power consumption. Moreover, hierarchical variational models are implemented to improve inference efficiency by utilizing shared statistical strength across different model layers. Precision in computational tasks is dynamically managed through techniques such as context-aware precision adjustment and mixed-precision quantization. Computational precision is continuously adapted based on external conditions such as vehicle speed and environmental complexity, as well as the importance of the current computational task. This strategy includes layer-specific quantization where critical network layers maintain higher precision, whereas less crucial layers operate with reduced bit-widths. The framework also features an energy-driven reconfiguration system that adjusts computational precision in response to dips below specific energy thresholds, ensuring sustained operation during low-power situations. Additionally, an adaptive floating-point format customizes the number of bits allocated for the exponent and mantissa to match the computational demands precisely.

These approximation strategies are integrated to optimize energy efficiency. The framework actively analyzes incoming tasks to determine the most suitable combination of approximation techniques based on the current energy constraints and task specifics. Real-time performance monitoring maintains a feedback loop that assesses the impact of these approximations on model performance, allowing for dynamic adjustments to ensure an optimal balance between energy efficiency and operational accuracy. Optimization simplifies the model complexity in scenarios where energy is critically low, prioritizing safety-critical functions to maintain essential system integrity. The goal of implementing these model approximation strategies in the framework is to lower the training and inference energy consumption of AI models within connected vehicles, enabling the deployment of advanced algorithms on resource-constrained edge devices.

7.3.4. Adaptive Model Deployment Strategies

The adaptive model deployment component in the framework aims to optimize model orchestration across distributed computing resources, which includes edge devices and cloud computing. This strategy uses advanced gradient-based techniques to achieve rapid convergence and efficient parallel computation, which is essential for handling dynamic environmental conditions and computational capabilities. The deployment strategy for partitioned models are discussed in the ARASEC and LAP-DTR mechanism in chapter 6.1 and chapter 6.2.

Gradient-Based Optimization Techniques: The framework uses advanced gradient methods including Enhanced Momentum Gradient Descent (EMGD) to accelerate the convergence of model training processes. EMGD incorporates a momentum term, aiding in navigating along the pertinent directions of the gradient. The mathematical representation is given by:

$$\psi^{(t+1)} = \gamma\psi^{(t)} + \nabla\text{Cost}(\Phi^{(t)}), \tag{7.1}$$

$$\Phi^{(t+1)} = \Phi^{(t)} - \eta\psi^{(t+1)}, \tag{7.2}$$

where $\psi^{(t)}$ is the velocity, γ is the momentum coefficient, $\nabla\text{Cost}(\Phi^{(t)})$ is the gradient of the cost function, and η is the learning rate.

Asynchronous Gradient Descent (AGD): This technique allows for asynchronous updates in a parallel computing setup, enhancing scalability and fault tolerance. AGD reduces the need for global synchronization, allowing each node to update based on locally computed gradients:

$$\Phi_{\text{global}}^{(t+1)} = \Phi_{\text{local}}^{(t)} - \eta\nabla\text{Cost}(\Phi_{\text{local}}^{(t)}),$$

where Φ_{global} and Φ_{local} represent the global and local model parameters, respectively.

Initialization and Iterative Refinement: Deployment begins with the initialization of model parameters and offload decisions, which are iteratively refined using performance and cost metrics. This process ensures optimal configuration based on the computational capabilities of processing units.

Constructing with Lookup Tables: Model components are mapped to hardware resources best suited to their needs using a lookup table approach. This method ensures that each model component is handled by the device with the required memory and computing resources, thus optimizing resource use and performance.

Edge Deployment: Utilizes EMGD to map models directly on multiple edge devices, taking advantage of fast convergence properties. AGD synchronizes model parameters across edge devices and the cloud, accommodating communication delays or failures effectively.

Cloud Deployment (AWS Lambda): This strategy uses AWS cloud computing instances capabilities to dynamically scale computational resources. EMGD enhances the training process, and AGD asynchronously updates model parameters across the distributed system, optimizing resource allocation and cost management.

Edge-Cloud Hybrid Deployment: This hybrid approach combines edge computing's low-latency and local data processing with the cloud's computational power. It starts with model training in the cloud using EMGD and distributes model parameters to edge devices via AGD, reducing data transfer requirements and enabling responsive model training. These adaptive deployment strategies ensure the framework meets the computational demands of complex AI models, managing orchestration, optimizing communication overhead, and maintaining rapid convergence across varied and dynamic network conditions. This approach is essential for deploying advanced AI models in resource-constrained environments, facilitating more efficient and scalable autonomous vehicle systems.

7.3.5. Energy-Aware Systems Using Reinforcement Learning

The above discussed components can independently be categorized as energy-efficient solutions. An approach to advances from energy-efficient to energy-aware solutions in connected vehicles using energy management and optimization approach is covered here. This transition involves moving from static, predefined energy-saving settings to dynamic, adaptive strategies that can respond in real-time to changing environmental conditions and operational demands. Reinforcement Learning (RL) offers a robust framework for facilitating this shift by enabling continuous learning and adaptation based on the system's interactions with its environment [176]. Reinforcement learning works on the principle of decision-making under uncertainty, where an agent learns to perform actions that maximize some notion of cumulative reward. In the context of connected vehicles, the RL agent interacts with a vehicular environment that is highly dynamic, characterized by continuous changes in driving conditions, network status, and energy availability [175, 177]. The agents objective is to develop a policy that optimizes energy usage without compromising the essential performance metrics such as safety, timeliness, and accuracy of navigational and operational tasks. Reinforcement Learning (RL) provides a robust framework for this shift, enabling continuous learning and adaptation based on the vehicle's interactions with its dynamic environment.

Reinforcement Learning for Dynamic Energy Management: RL operates under the principle of maximizing a cumulative reward, with the RL agent developing policies that optimize energy use while maintaining essential performance metrics such as safety and timeliness. The RL framework consists of:

State Space: This includes necessary data about the operational status, including battery levels, processor and memory utilization, latency and network conditions required for making adaptive decisions.

Action Space: Actions include adjusting computational resource speeds (CPU/GPU), and decisions on task offloading to optimize energy consumption efficiently.

Reward Function: Designed to balance high performance with low energy consumption, this function penalizes excessive energy use and rewards reductions in energy consumption when a model partition, resource allocation and an approximation strategy is used. The implementation steps are:

Training the RL Agent: Initially, the agent is trained using Deep Q-Network within a simulated environment replicating operational scenarios with diverse memory and computational load. This phase allows the agent to learn optimal policies without real-world effects, using simulated data (model training and evaluation based on batch sizes) to predict energy requirements accurately.

Real-Time Learning and Adaptation: Post-deployment, the RL agent continually refines its policies based on incoming real-world data. This ongoing learning process ensures that the system remains adaptive to changing conditions and can optimize decisions for energy management dynamically.

Integration with Task: The trained RL agent's policies are integrated into the operational applications, for e.g., a perception task, enabling real-time adjustments to the energy management strategies based on the vehicle's current state and its environment.

Model Approximation: These strategies reduce computational demands using probabilistic kernel applications or sparse activation maps, defined as:

$$E_{approx} = \sum_{i=1}^{n} E_{orig,i} - E_{approx,i}$$

where $E_{orig,i}$ and $E_{approx,i}$ represent the original and reduced energy consumption for each task.

Model Partitioning: This involves distributing computation to optimize energy usage across vehicle, edge-cloud layers:

$$E_{part} = E_{local} + \sum_{j=1}^{m} E_{trans,j} + E_{remote,j}$$

Here E_{local}, $E_{trans,j}$, and $E_{remote,j}$ are used for local computations and data transmission energies.

Model Deployment Mechanisms: Both synchronous and asynchronous mechanisms are used to process data, influencing overall energy consumption:

$$E_{deploy} = \alpha E_{sync} + (1-\alpha)E_{async}$$

Here α refers to the operations using synchronous methods. Utilizing RL in this capacity enhances overall energy management in connected vehicles, dynamically adjusting operational strategies to maintain optimal performance levels while minimizing energy consumption.

Integration of Lookup Table and Adaptive Mechanisms: The Lookup table acts as the initial knowledge base for the RL agent, allowing early decision-making, which the agent refines through ongoing environmental interactions.

Adaptive Mechanisms: As the RL agent gains experience, reliance on the lookup table decreases. A switching mechanism balances the use between the static lookup table and the dynamically learned policies, enhancing adaptability.

Predictive Models: An LSTM model predicts future energy consumption based on current trends and workload, which informs the RL agent's decisions, enhancing its capacity to manage energy dynamically.

Multi-Objective Optimization: As the energy consumption of framework components is balanced with multiple model performance metrics, the framework also considers multiple objectives, such as energy consumption, accuracy, latency, throughput, sensitivity metrics and speed-up (training and inference). Techniques including weighted grey relational analysis, are used to select optimal configurations, ensuring efficiency across multiple dimensions.

This approach aligns with sustainability goals and improves the deployment strategy of advanced AI applications in energy-limited scenarios for autonomous and efficient vehicular technologies. By implementing these strategies, our framework improves the energy efficiency of connected autonomous vehicles to operate efficiently and sustainably under varying operational conditions.

7.3.6. Integration of Proposed Components

Integrating the proposed components: resource allocation & management, approximation scheme, adaptive model deployment, and energy management improves its capabilities from energy-efficient computing to energy-aware computing. The resource allocation-management module interfaces with the previous heterogeneous-aware scheduler, optimizing the distribution of computational tasks across available resources while considering energy constraints. The approximation scheme component works in coordination with the computing unit platform, dynamically adjusting the precision of operations based on the capabilities of each computing unit and current energy levels. Adaptive model deployment uses the vehicle-edge coordinator to intelligently partition and distribute AI models across vehicle and edge resources, minimizing data transfer and energy consumption. Finally, the energy management module integrates with the task profiler, continuously monitoring energy usage and providing output to components for real-time optimization. This integration enables the framework to make energy-aware decisions at different stages of task execution, from initial profiling to final deployment, resulting in a more efficient and adaptable system for vehicle-edge services.

7.3.7. Memory Estimation for GPU Usage for Inference

To estimate the GPU memory required for training or inference of a model, we consider three main components: parameter memory, activation memory, and framework overhead. The calculations for these components are detailed below.

1) Parameter Memory: The memory required to store model parameters is calculated as:

$$\text{Memory}_{\text{params}} = \frac{N_{\text{params}} \times \text{Bit\_width}_{\text{param}}}{8}$$

where:

- N_{params} is the total number of parameters in the model.
- $\text{Bit\_width}_{\text{param}}$ is the bit width of each parameter (e.g., 16 for fp16 or 32 for fp32).

For instance, a model with 7×10^9 parameters at 16-bit precision requires:

$$\text{Memory}_{\text{params}} = \frac{7 \times 10^9 \times 16}{8} = 14\,\text{GB}$$

2) Activation Memory: Activation memory depends on the activation tensors for each sample and scales with the batch size. For each layer i, activation memory is:

$$\text{Memory}_{\text{act\_layer\_i}} = \frac{N_{\text{elements\_activation\_layer\_i}} \times \text{Bit\_width}_{\text{activation}}}{8}$$

The largest activation tensor across all layers is considered:

$$\text{Max\_Memory}_{\text{act\_per\_sample}} = \max(\text{Memory}_{\text{act\_layer\_i}})$$

The total activation memory for a batch size is:

$$\text{Memory}_{\text{activations}} = \text{Batch\_size} \times \sum_{i=1}^{L} \text{Memory}_{\text{act\_per\_sample\_layer\_i}}$$

where L is the total number of layers.

Transformer models have attention layers with memory proportional to:

$$M_{\text{attention}} = B \times T^2 \times D$$

where:

- B is batch size,
- T is sequence length,
- D is hidden dimension.

3) Total GPU Memory
The total GPU memory required is approximately:

$$\text{Memory}_{\text{total}} = \text{Memory}_{\text{params}} + \text{Memory}_{\text{activations}} + \text{Memory}_{\text{gradients}} + \text{Overhead}$$

where:

- Gradients typically require memory equal to $2 \times \text{Memory}_{\text{params}}$, accounting for gradients and optimizer states (e.g., Adam optimizer).
- Overhead accounts for framework-specific usage (PyTorch or TensorFlow), around $15\% - 20\%$ of the total memory graph of the model.

For a transformer model with:

- $N_{\text{params}} = 1.5 \times 10^9$,
- Batch size $B = 32$,
- Sequence length $T = 512$,
- Hidden dimension $D = 1024$,
- Bit width (16 bits),

we calculate:
Parameter Memory:

$$\text{Memory}_{\text{params}} = \frac{1.5 \times 10^9 \times 16}{8} = 3\,\text{GB}$$

Activation Memory (Attention):

$$M_{\text{attention}} = 32 \times 512^2 \times 1024$$

Gradients: $2 \times 3\,\text{GB} = 6\,\text{GB}$.
Overhead: Approximately 10%–20% of the total calculated memory.

7.4. Experimental Setup

The experimental setup (hardware configurations, model selections, datasets, evaluation metrics, and energy measurement profiling) follows the same procedure described in Section 6.2.4.

7.5. Evaluation of the Framework

The effectiveness of the proposed energy-aware framework is assessed through a series of experiments designed to test its performance under varying data rates and computational conditions. This evaluation aims to show the framework's adaptability, efficiency, and responsiveness when deployed across used edge devices.

Data Rate Variability: To simulate real-world usage scenarios, responsiveness and adaptability, we vary the data rate for image batches processed by the Vision Transformer (ViT) models. Three different batch sizes are used: 32, 64, and 128. This variation helps us understand how changes in data input rates affect the latency, throughput, and energy consumption of the system.

Hardware and Models: The tests are conducted using a heterogeneous set of edge devices, as previously described, to cover wide computing capabilities. We use several ViT models suitable for edge deployment, including EdgeViT and TinyViT, to explore the trade-offs between computational demand and performance across different hardware setups.

Dataset: The OPV2V dataset is utilized to provide a consistent testing ground for our models, ensuring that the evaluation reflects performance under conditions that mimic real-world autonomous driving tasks.

Latency: Measured as the time taken from input feature map to output feature map, latency is a critical measure for assessing vision application's practicality, particularly in autonomous driving where timely data processing is necessary.

Throughput: We evaluate the number of images processed per unit time under each batch setting to gauge the system's efficiency and ability to handle high volumes of data.

Energy Consumption: Using tools like Tegrastats, we precisely measure the energy used during the inference process on each device. This metric is crucial for validating the energy-efficiency claims of the proposed framework.

For each combination of model, device, and data rate, we capture detailed metrics on latency, throughput, and energy consumption. These measurements allow us to draw comparisons and identify optimal configurations for different operational scenarios.

Analytical Tools and Techniques: We use statistical methods to ensure the reliability of our observations and to perform comparative analysis across different setups.

Visualization: Charts and graphs are used to visually represent the data, making it easier to interpret the effects of different batch sizes and hardware configurations on the performance metrics.

Overall evaluation is complemented by summarizing latency, throughput, mAP and energy consumption. The analysis includes the impact of batch size on the performance metrics. The trade-offs between latency, throughput, and energy consumption for different models and hardware. How effectively does the framework adapt to varying operational demands and data rates?

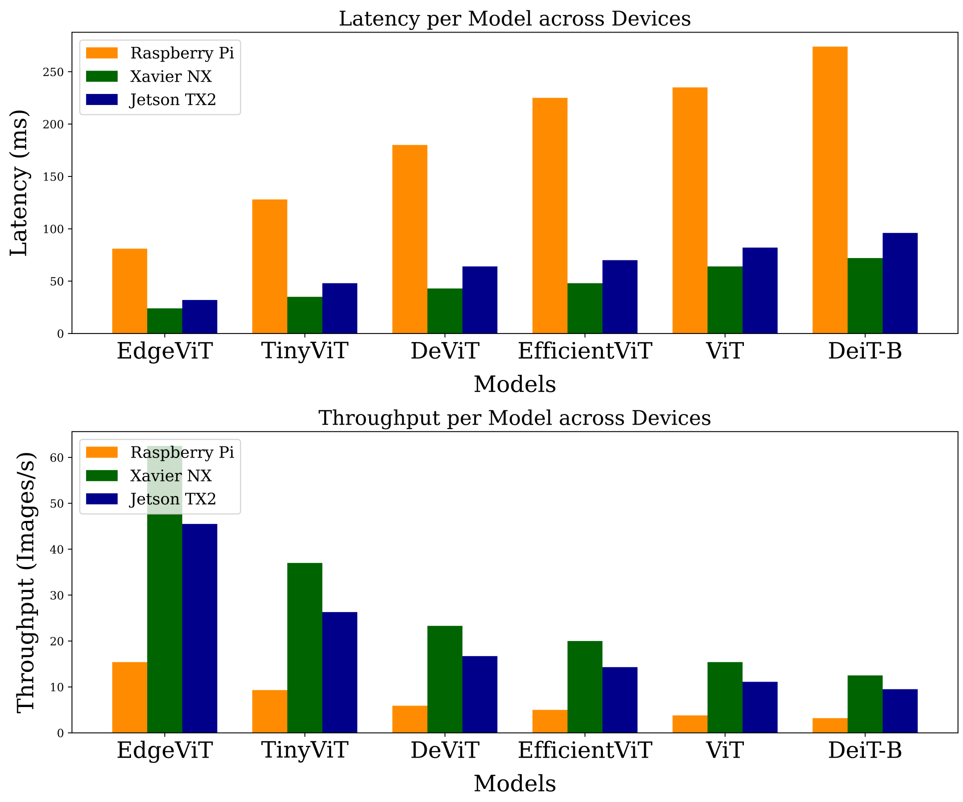


Figure 7.2: Latency and Throughput Comparison for a batch of 32

7.5.1. Detailed Analysis of Latency and Throughput

This section evaluates latency and throughput across various Vision Transformer (ViT) models on different hardware devices for batch sizes 32, 64, 128, and 256. These metrics are critical for assessing the proposed energy-aware methods' performance scalability and efficiency in practical applications.

For model performance metrics on devices, i.e. latency and throughput, the Raspberry Pi shows higher latency values (up to 274 ms for DeiT-B), reflecting its limited processing capabilities as shown in Figure 7.2. Xavier NX and Jetson TX2 show lower latency (72 ms and 96 ms for DeiT-B, respectively), highlighting their robust computational power. **Throughput:** With high latency, the Raspberry Pi manages throughputs above 3 images/sec for simpler models. Xavier NX, because of processing power, has the highest throughput of 62.5 images/sec for EdgeViT.

As shown in Figure 7.3, for a batch size 64, latency value increase with a factor of 2 across all devices, with Raspberry Pi reaching up to 562 ms for DeiT-B. Xavier NX and Jetson TX2 keep latency under 200 ms for all models. Throughput values are decreased with high batch size for Raspberry Pi and TX2. However, Xavier NX maintains a throughput

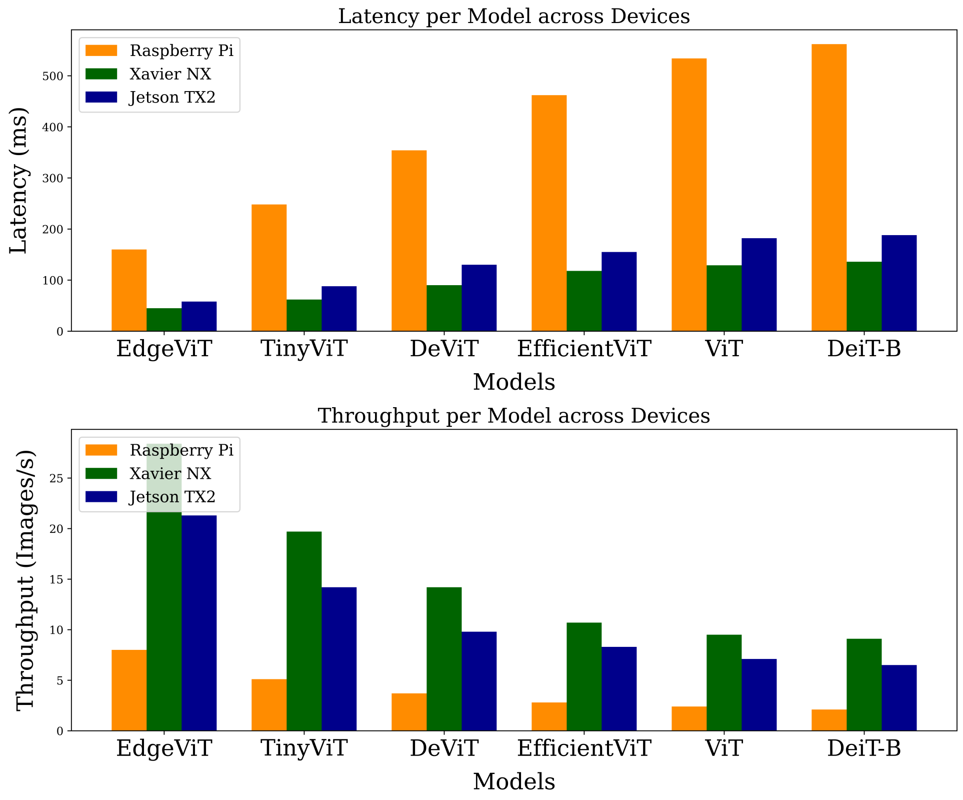


Figure 7.3: Latency and throughput comparison for a batch of 64.

above 9 images/sec across all models, indicating good scalability. For a batch size of 128, as shown in Figure 7.4, increases in latency value is measured, with Raspberry Pi exceeding 1 second for some models. Both Xavier NX and Jetson TX2 show lower, more manageable latencies. Further reductions in throughput values are observed, with Xavier NX and Jetson TX2 performing robustly, maintaining rates above 4 and 3.6 images/sec, respectively, for the most demanding models, showing the efficient GPU utilization by these devices.

Similarly, for a batch size of 256 (as shown in Figure 7.5, there are significant increases in latency, with Raspberry Pi exceeding 2 seconds for complex models, likely impacting real-time application feasibility. Xavier NX and Jetson TX2 experience increased latency but remain within more acceptable ranges. The throughput decreases significantly across all devices, with Xavier NX maintaining the highest throughput, showing its ability to handle larger computational loads with manageable resources. The latency and throughput analysis across different batch sizes highlights the varying capabilities of hardware devices in managing the computational demands of advanced Vision Transformer models. Xavier NX generally provides the best balance of latency and throughput, making it suitable for more computationally intensive applications, whereas Raspberry

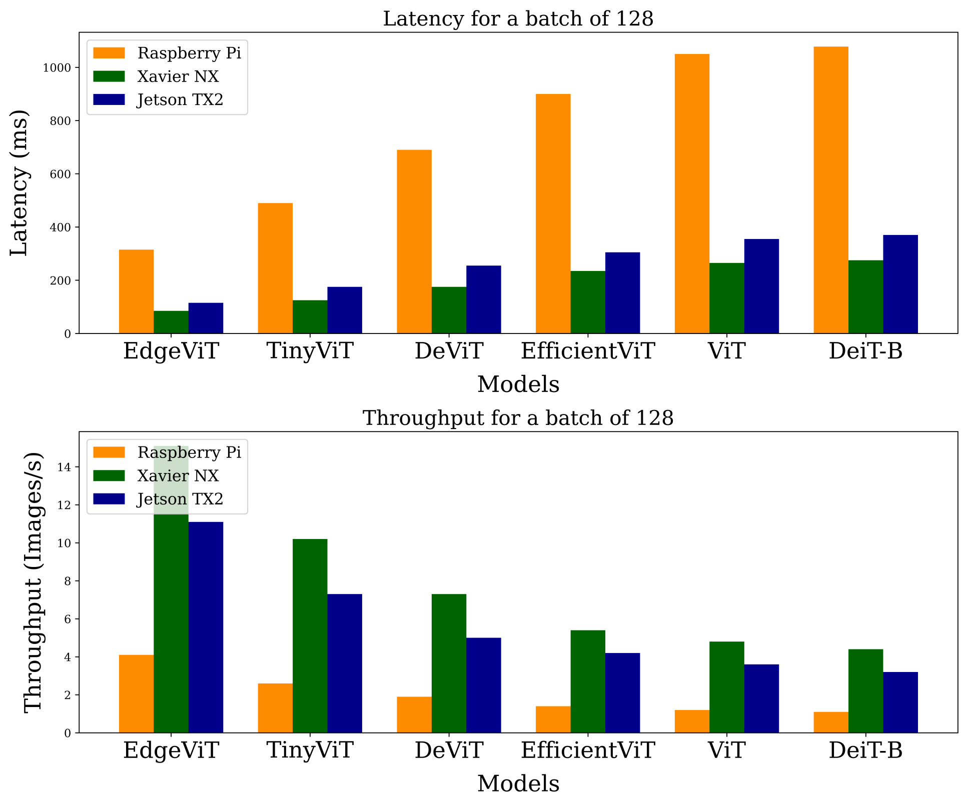


Figure 7.4: Latency and throughput comparison for a batch of 128.

Pi lacks performance with higher batch sizes, emphasizing the importance of hardware considerations in deployment scenarios. These insights are essential for directing future optimizations and selecting appropriate hardware for deploying deep learning models in edge environments.

7.5.2. Energy Efficiency and Accuracy Analysis

The allocation of energy consumption across CPU, memory, and GPU components for various Vision Transformer (ViT) models at different batch sizes provides key insights into the computational efficiency and operational dynamics of these models. This analysis helps in optimizing the deployment strategies on edge devices by understanding where energy optimization can be most effective.

For a batch size 32, as shown in Figure 7.6, CPU usage varies from 20% to 25%, indicating moderate CPU usage across the models. Memory consumption accounts for 22% to 28% of the total energy, which shows the importance of memory (RAM) in managing data flow and storage during vision transformer model operations. The GPU, crucial for computation, consumes the majority of the energy, ranging from 51% to 57%. This

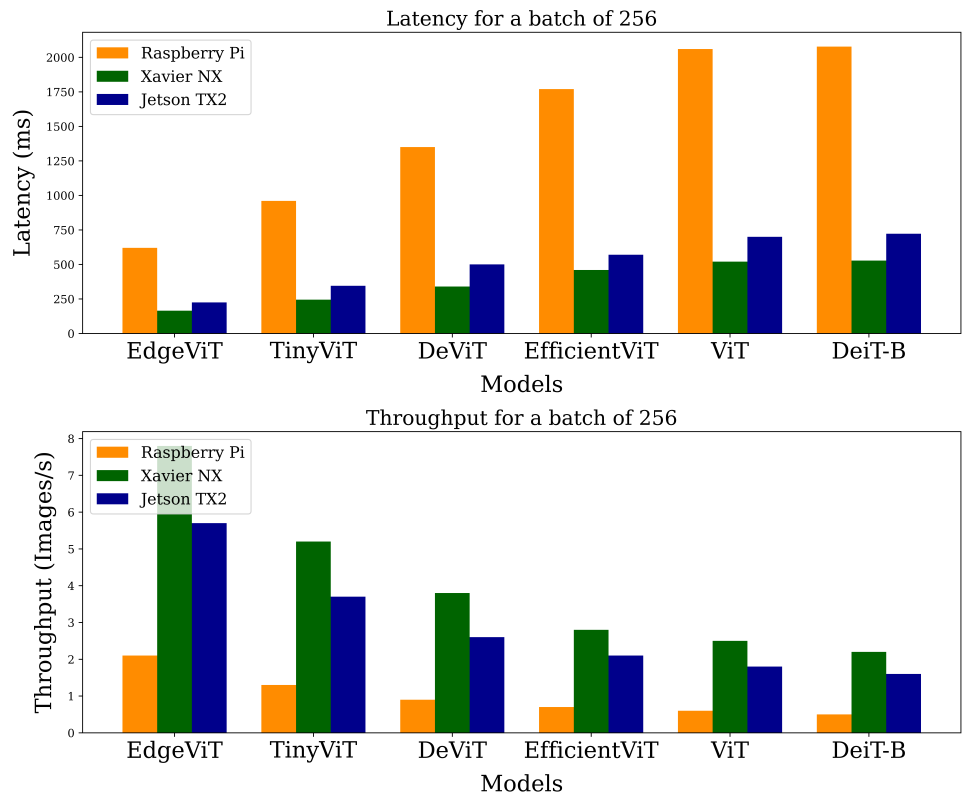


Figure 7.5: Latency and throughput comparison for a batch of 256.

highlights the GPU's dominant role in processing the complex architectures of vision transformers. As the batch size increases to 64 (also shown in Figure 7.7), there is a decrease in CPU usage, for 18% to 23% of the energy. This suggests a relative decrease in CPU-bound tasks. However, memory usage increases to 24% to 30%, reflecting the high demand of input feature map for data handling with larger batches. GPU usage remains relatively stable, showing the approximate strategies efficiency in GPUs computational process, maintaining the energy consumption of 52% to 57%.

For the batch size of 128, as shown in Figure 7.8, there is a continuous decline in CPU energy consumption (16% to 21%), suggesting more efficient CPU utilization or potential CPU resource saturation with larger data sets. Memory usage further increases from 26% to 32%, supporting the trend of increased memory requirements for larger batch operations. GPU energy consumption remains high, ranging from 52% to 57%, indicating that with the increased batch sizes, GPUs are primarily responsible for the output feature map computations. For batch size of 256, CPU energy usage is lowest, ranging from 14% to 19%, indicating optimized batch processing efficiencies by TensorRT. Memory consumption increases at 28% to 34%, the highest among all batch sizes, as more data is cached to maintain operational throughput. The GPU continues to bear the major share

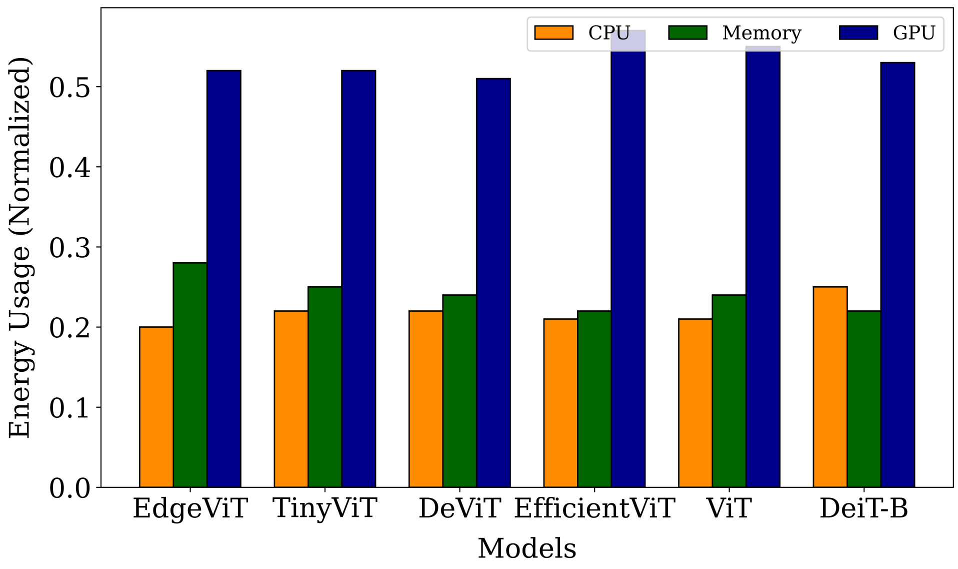


Figure 7.6: Energy Comparison for models on Jetson NX for a batch of 32.

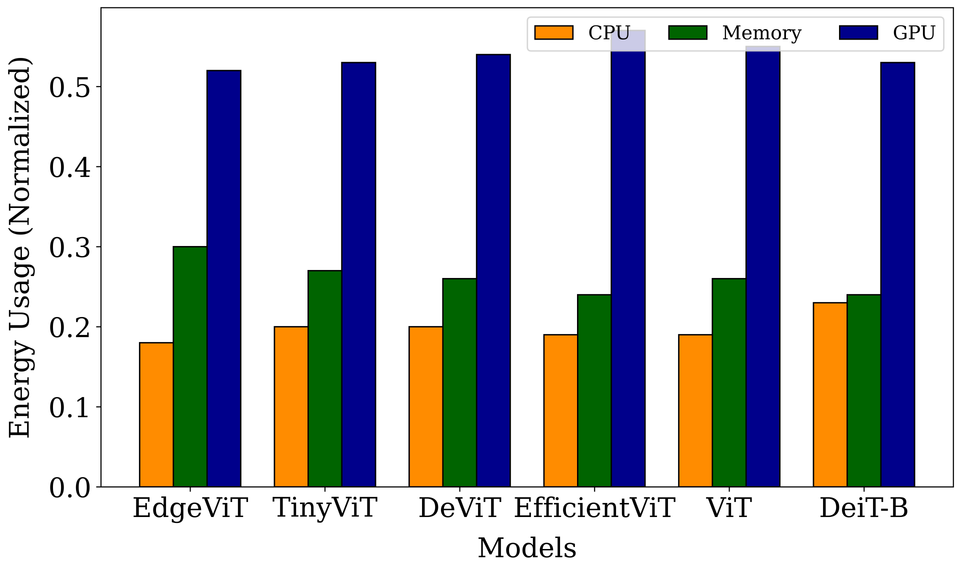


Figure 7.7: Energy comparison for models on Jetson NX for a batch of 64.

of computational load with 52% to 57% of energy use, executing complex deep learning operations efficiently, s shown in Figure 7.9.

Overall this analysis of energy consumption across components with increasing batch

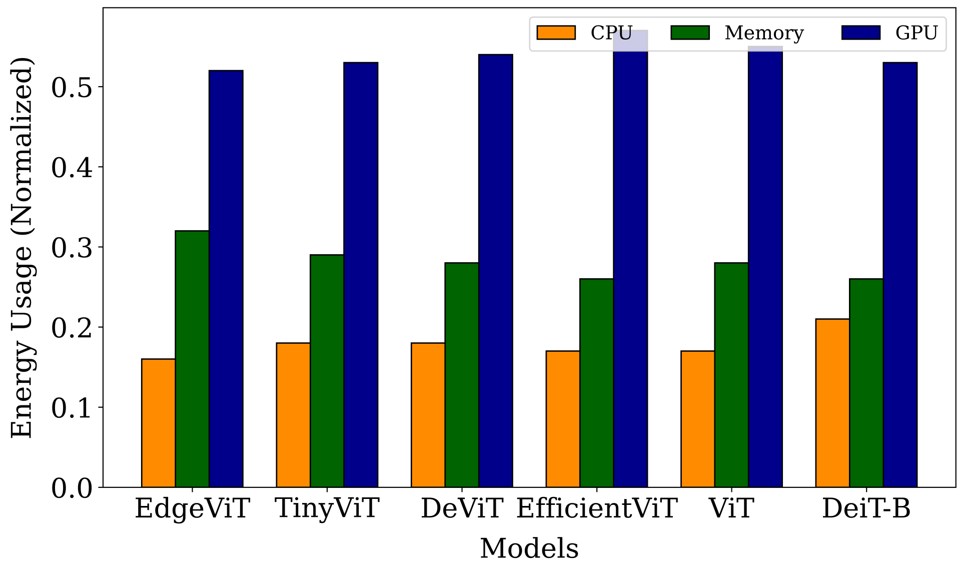


Figure 7.8: Energy Comparison for models on Jetson NX for a batch of 128.

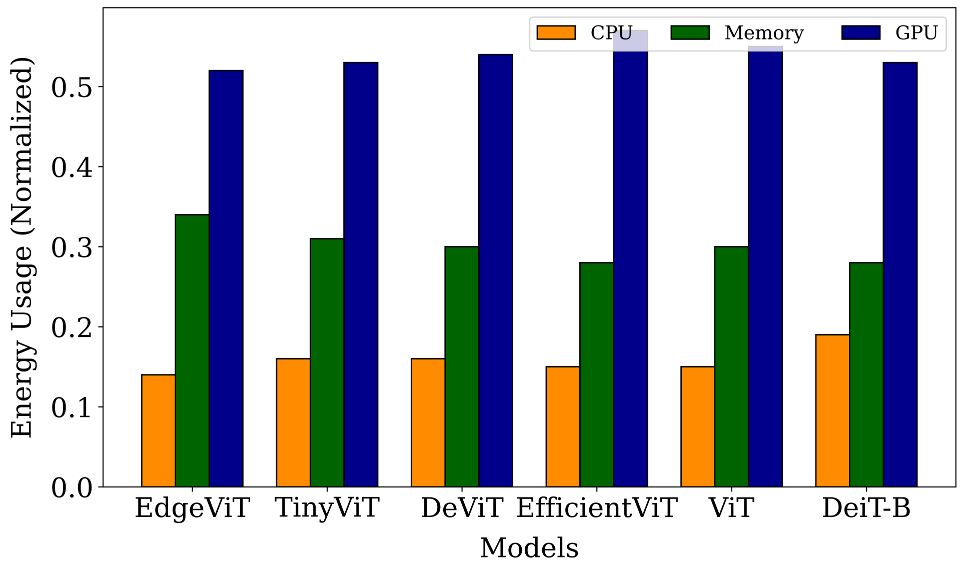


Figure 7.9: Energy Comparison for models on Jetson NX for a batch of 256.

sizes shows a shift from CPU to more GPU and memory-intensive operations as model complexity and batch size increase. Such insights are critical for deploying Vision Transformer models in edge computing environments and prioritizing computational tasks (e.g., fine-tuning or over-the-air update) where energy efficiency is an important factor. The consistent high energy usage by GPUs across all batch sizes reinforces the need for energy-efficient GPU technologies and optimization techniques to enhance the sustainability of deploying advanced AI models. Our study evaluates the performance and energy efficiency of various Vision Transformer (ViT) models across different batch sizes. The metrics compared include mean average precision (mAP), latency, and energy consumption, comparing a centralized approach with our proposed method. As batch sizes increase from 32 to 256, a progressive increase in latency is observed across all models, which aligns with the increased computational demands. Our proposed method consistently shows reduced latency and energy consumption compared to the centralized approach, showing its efficacy in optimizing computational resources.

Table 7.1: Baseline and Proposed Method Comparison (Batch Size: 32).

Models	Methods	mAP (%)	Latency	Energy (mJ)
EdgeViT	Central	57.3	4.25	370.6
	AxC-Energy	55.6	3.88	290.7
TinyViT	Central	55.7	3.70	430.6
	AxC-Energy	51.1	3.27	367.5
EfficientViT	Central	53.9	3.86	462.7
	AxC-Energy	51.7	3.34	381.6
ViT	Central	49.7	4.73	493.6
	AxC-Energy	42.5	3.02	402.5
DeiT-B	Central	46.6	5.05	432.0
	AxC-Energy	40.2	3.54	341.4

7.5.3. AxCEnergy Comparison with Baseline

To show the efficiency of the proposed AxCEnergy framework, we compare it with the baseline (centralized) method. An ablation test to measure mAP, latency and energy for a single forward pass with different batch sizes is conducted on Xavier NX for AxCEnergy. To capture performance for all the models using the centralized method, V100 Nvidia GPU is used (detailed specifications of devices are covered in chapter 6.

As shown in Table 7.1, for **EdgeViT**, there is a reduction in latency from 4.25 ms to 3.88 ms and in energy consumption from 370.6 mJ to 290.7 mJ, showing significant efficiency gains with our method. Similarly, **DeiT-B** shows the most substantial energy efficiency improvement, reducing consumption from 432.0 mJ to 341.4 mJ. For a batch Size 64 covered in Table 7.2, **EfficientViT** shows improvements, with latency decreasing from 4.06 ms to 3.54 ms, showing the method's capability to enhance performance without extensive power usage. **TinyViT** achieves a good balance between performance and energy, emphasizing its suitability for resource-constrained environments.

For a higher batch size 128 in Table 7.3, the **ViT** model experiences a significant decrease in latency from 5.13 ms to 3.42 ms under our method, showing higher adaptability in managing computationally intensive tasks efficiently. Moderate increases in latency and energy usage across models affirm the scalability of our method. For a batch

Table 7.2: Baseline and Proposed Method Comparison (batch size: 64).

Models	Methods	mAP (%)	Latency	Energy (mJ)
EdgeViT	Central	57.1	4.45	380.6
	AxC-Energy	55.4	4.08	300.7
TinyViT	Central	55.5	3.90	440.6
	AxC-Energy	51.0	3.47	375.5
EfficientViT	Central	53.7	4.06	472.7
	AxC-Energy	51.5	3.54	391.6
ViT	Central	49.5	4.93	503.6
	AxC-Energy	42.3	3.22	412.5
DeiT-B	Central	46.4	5.25	442.0
	AxC-Energy	40.0	3.74	351.4

Table 7.3: Baseline and Proposed Method Comparison (Batch Size: 128).

Models	Methods	mAP (%)	Latency	Energy (mJ)
EdgeViT	Central	56.9	4.65	390.6
	AxC-Energy	55.2	4.28	310.7
TinyViT	Central	55.3	4.10	450.6
	AxC-Energy	50.9	3.67	383.5
EfficientViT	Central	53.5	4.26	482.7
	AxC-Energy	51.3	3.74	401.6
ViT	Central	49.3	5.13	513.6
	AxC-Energy	42.1	3.42	422.5
DeiT-B	Central	46.2	5.45	452.0
	AxC-Energy	39.8	3.94	361.4

size 256 as shown in Table 7.4, **EdgeViT** and **EfficientViT** show better efficiency, particularly EdgeViT, which achieves the lowest energy consumption at 320.7 mJ among the high-performance models, indicating an optimal balance between speed and energy use. **DeiT-B** consistently shows improvements in both latency and energy consumption, suggesting exceptional compatibility with the proposed method. Overall evaluation from the inference test shows that the proposed method significantly reduces latency and energy consumption across various ViT models, offering scalable performance as batch sizes increase. The reductions in latency and energy are especially pronounced in high-performance models, enhancing operational efficiency and sustainability in real-world applications. These findings provide a strong foundation for future optimizations, targeting enhanced energy management while maintaining high-performance levels.

7.6. Conclusion

The existing energy-efficient edge frameworks [70–72] address the challenges of memory and computation availability, communication in connected vehicles, and cloud and edge computing approaches to enable connected vehicular services. However, they specifically lack model partition, resource-aware allocation, adaptive deployment, and energy-

Table 7.4: Baseline and Proposed Method Comparison (Batch Size: 256).

Models	Methods	mAP (%)	Latency	Energy (mJ)
EdgeViT	Central	56.7	4.85	400.6
	AxC-Energy	55.0	4.48	320.7
TinyViT	Central	55.1	4.30	460.6
	AxC-Energy	50.7	3.87	391.5
EfficientViT	Central	53.3	4.46	492.7
	AxC-Energy	51.1	3.94	411.6
ViT	Central	49.1	5.33	523.6
	AxC-Energy	41.9	3.62	432.5
DeiT-B	Central	46.0	5.65	462.0
	AxC-Energy	39.6	4.14	371.4

aware approximation. These requirements resulted from research questions 1, 2, and 3 and have been thoroughly discussed in chapters 4, 5, and 6, respectively. This chapter proposes an adaptive framework that includes model partition mechanisms, adaptive model deployment, and model approximation strategies to balance energy efficiency with model performance and accuracy using an energy-aware approach. The discussion covered proposed components, their integration, and testing using vision models to answer the mentioned research questions, which shows the potential for onboard energy savings and balanced performance in the vehicular ecosystem. Future work can test the framework across heterogeneous edge devices, including accelerators and multi-modalities, to validate its adaptability and effectiveness in diverse operational environments. Additionally, the scope of vehicular applications can be expanded to include more complex computational tasks with high data volumes, such as high-definition video streaming, where some computational processes are offloaded and shared with cloud environments, which can help explore the balance between edge and cloud computing, aiming to optimize resource allocation and further reduce the energy footprint of connected vehicular systems.

8 Conclusion

Adopting AI and ML models within connected vehicles and cyber-physical systems requires large-volume data processing, data transfer and high-performance computing. Such adaptation provides application-level advantages for deployment but also presents system-level challenges. These ML-supported applications integrate complex computational tasks ranging from perception to real-time decision-making and continuous data transfer with other vehicles and infrastructure devices. This process requires optimized computing and efficient data processing capabilities, supported by complex algorithms and models like convolution and deep neural networks and vision transformers within computing frameworks. However, scalable deployment of these models to handle tasks like traffic monitoring, segmentation, detection, localization and mapping introduces challenges related to energy consumption, particularly in scenarios where computing units operate on limited memory and processor resources, including power sources. For example, processing high-definition (HD) maps and real-time video streams are computationally intensive tasks that impact the vehicle's battery life and operational efficiency. These computational demands are further increased with the need for constant connectivity and data exchange involving frequent transmissions of data and model weights within the vehicular ecosystem, requiring energy resources. This chapter summarizes challenges of high energy consumption from the data-intensive connected vehicle services, then discusses the research problems covered in chapter 1 and research contributions, and lastly discusses the limitations and future research directions.

8.1. Overview

Edge AI addresses latency and bandwidth challenges by processing sensed data close to its source, which is important for connected vehicle applications requiring constant connectivity and high data processing capabilities. Edge AI practices reduce dependency on centralized systems but introduce challenges in managing memory, computing power, and performance from resource-constrained devices. Optimization and profiling of edge devices and systems for low power consumption without compromising computational performance are necessary, as connected vehicles and respective services integrate complex algorithms. Under limited resources, these units must perform advanced computations, such as object recognition and decision-making. Energy-aware models that dynamically balance computational requirements (e.g., CPU, GPU, memory) based on real-time energy availability and task prioritization have been developed to address these tradeoffs. Such models use energy-efficient algorithms and optimizations that reduce computational overhead while maintaining accuracy and model performance metrics. Techniques such as energy-aware model partitions and adaptive deployment distribute computational tasks across the vehicle's edge network and cloud infrastructure efficiently, while approximation strategies simplify operations, further reducing energy consumption. These approaches within the vehicular ecosystem balance the computational capabilities required for advanced autonomous applications to be energy-aware, aligning with the goal of sustainable practices in AI research and development. **Therefore, this research identified the key design requirements for energy-efficient computing at the Edge, and based on these design requirements, developed an energy-aware adaptive framework which balances performance and energy consumption for resource-constrained edge devices in connected vehicle systems.**

This thesis used 'design science research methodology' to explore the design requirements and identify the components needed by answering the research questions for the development of an energy-aware adaptive framework. Research exploration in the context of connected vehicle applications within the thesis primarily addressed the dual challenges of high-performance computing requirements and associated energy consumption. The summary of research contributions corresponding to the four questions is as follows.

Research Question 1: *What are the technical and operational requirements to achieve energy efficiency in data-intensive connected and collaborative vehicular services involving multiple stakeholders?*

Contribution 1):

Exploration provided technical and operational requirements for energy-efficient operations within connected vehicle networks. It laid the foundation for thinking of solutions beyond standard compression mechanisms and exploring the application of advanced approximation techniques, model partition mechanisms and resource allocation mechanisms.

Research Question 2: *How can the energy and performance tradeoff be balanced for vehicle-edge-cloud computing?*

Contribution 2):

This work identified software-level approximation computing opportunities at edge devices for the classic and new generation of vision models that significantly decrease energy usage while maintaining computational efficiency. These strategies are important for developing adaptive systems that accurately balance energy and performance needs.

Research Question 3: *How to achieve an energy-aware adaptive framework from energy-efficient approximation schemes?*

Contribution 3):

Integrating energy-efficient components into an adaptive framework with reinforcement learning algorithms, the research created a prototype framework capable of adjusting its functions based on current available computing demands, energy conditions and operational requirements.

Research Question 4: *Can the framework effectively balance the trade-off between energy efficiency and performance in vehicle-edge-cloud computing scenarios?*

Contribution 4):

The application and thorough evaluation of the framework shows its capacity to sustain a balance between reducing energy consumption and maintaining high performance, validating its effectiveness and utility.

The research exploration in this thesis has contributed to a new understanding of the field of energy-aware computing within connected vehicular systems. By addressing the research questions, this thesis aims to fill existing gaps and advanced technical knowledge of training and deploying complex AI models efficiently with limited computing resources. This research has contributed to energy-efficient mechanisms and transitioning to energy-aware computing in connected vehicles by developing frameworks and models that address both performance and energy consumption.

8.2. Technical Contributions

The technical contributions of this thesis are within the scope of multiple energy-efficient strategies and an energy-aware framework integrating these strategies (Figure 7.1) for efficient training and inference of connected vehicle services, applications and systems. The research particularly addresses the gaps in resource allocation, model partitioning, and computational optimization strategies for collaborative applications that can be deployed using heterogeneous edge devices. The technical contributions directly corresponds to the artefacts as outlined in Table 2.1 from chapter 2, thereby ensuring domain-specific methods and approaches which can be summarized as follows:

1) Energy-Efficient AI Pipeline: An edge AI processing pipeline has been developed and evaluated for energy-efficient vehicular services:

- **Implementation and Testing:** Development of an edge AI processing pipeline, which directly relates to the *System Design* artefact by integrating model partitioning and approximation schemes for energy-efficient vehicular services
- **Effective integration and validation** of this pipeline within a broad adaptive framework, enabling energy savings and performance efficacy, correlating with the *Instantiation* artefact.

2) Bias Identification and Mitigation in AI Models Progress in identifying and mitigating bias in AI models has improved the fairness and accuracy of these systems:

- **Advanced bias detection techniques**, aligning with the *Method* artefact by using model behavioural metrics for improving fairness and accuracy across diverse datasets.

- **Development of metric-specific learning strategies**, enhancing generalization and reducing biases, which builds upon the *Algorithm* artifact.

3) Resource Allocation and Model Partition Strategies The thesis introduces methods for dynamic resource allocation and model partitioning based on model profiling and look-up-table which improves the operational efficiency of edge computing frameworks:

- **Resource Allocation Mechanisms**, optimized to distribute computational and energy resources efficiently, supports the *Method* artefact by ensuring responsive and efficient system operation.

- **Model Partitioning Approaches**, allowing for efficient deployment of large models across heterogeneous computing environments, derived from the *Algorithm* and *System Design* artefacts.

4) Edge AI Framework for CAV: This work has developed an energy-aware computing framework that utilizes edge intelligence to optimize the deployment of data and compute intensive vehicular services on edge devices. The adaptive framework developed has been evaluated to validate its effectiveness in real-world scenarios. The framework's components and contributions include:

- **Model Approximation Strategies:** Focused on energy-efficient model training and inference through advanced techniques like reduced precision, approximate multipliers, variational inference with post-training quantization and energy-aware training, ensuring high computational efficiency.

- **Model Partitioning Techniques:** Strategies to distribute the computational load from CNN, DNN and ViTs optimally across the edge and cloud resources using look-up table mechanisms. The proposed techniques when combined with approximation strategies balance energy consumption with computational specific metrics, validating its effectiveness and utility in operational scenarios.

- **Dynamic Adaptive Deployment Strategies:** Techniques that adapt computational accuracy based on real-time computing and energy requirements. The strategies are adaptive and scalable across varied computing and application environment.

The technical contributions outlined above are interrelated as mentioned in previous chapters, collectively forming a unified approach to energy-aware and efficient AI deployment in connected vehicle environments. The development of an energy-efficient AI pipeline establishes the foundational infrastructure, enabling subsequent advances in model bias identification, resource allocation, and partitioning. Bias mitigation techniques directly enhance the pipelines reliability and fairness, while advanced resource allocation and model partition strategies ensure that computational workloads are distributed optimally across heterogeneous edge platforms. These foundational strategies are further embedded and tested within the adaptive Edge AI framework, where model approximation methods, dynamic deployment mechanisms, and partitioning techniques operate in concert to maximize energy efficiency and performance. By integrating these individual contributions into a comprehensive framework, the research delivers a solution that addresses algorithmic robustness, operational scalability, and system-level efficiency demonstrating how each element reinforces and complements the others to meet the complex requirements of collaborative, data-intensive vehicular services.

8.3. Societal Impact

This research also impacts societal advancements and progress by aligning with international sustainability goals aimed at reducing climate and environmental impact caused by technological innovation. The United Nations Sustainable Development Goals, SDG 9 (Industry, Innovation, and Infrastructure) [47], SDG 11 (Sustainable Cities and Communities) [48], highlights the need to integrate energy-efficiency as a metric and factor in industrial and urban development. By optimizing computational efficiency and reducing energy consumption in connected vehicle systems, this thesis supports the broader agenda of improving sustainable industrialization for connected intelligent transportation systems, which are also expected to be integrated with smart cities. By improving computational efficiency and reducing energy consumption in connected vehicle systems, this work supports sustainable AI practices and aligns with the European Union's policies promoting reduced ICT-related energy consumption [178]. The energy-aware framework developed in this thesis helps in addressing these challenges by providing strategies for AI model training and inference, which can be adapted for other applications, including connected vehicles, to further help in reducing the environmental footprint of advanced computing systems, thus contributing to sustainable development in technology-intensive sectors. The research methods, algorithms, and technical implementation discussed help make a measurable impact on reducing the need for and larger dependency on high-performance computing, resulting in power consumption and associated carbon footprint, and further aligning with global efforts to promote environmental sustainability in technological developments.

8.4. Limitations and Future Research Directions

The proposed framework for energy-aware systems in Edge AI presents several technical and operational challenges. These limitations highlight the need for improvements in algorithm development, hardware capabilities, and system designs to enhance energy-aware systems' adaptability, efficiency, and safety in connected vehicles. Here we detail the key limitations:

- **Safety-Critical Applications:** The challenge of applying approximate computing in safety-critical applications directly arises from the requirements-gathering and literature review phases of this research. While the systematic literature review focused on identifying energy-efficient approaches suitable for data-intensive vehicular services, it also highlighted the trade-off between error tolerance and the strict demands of safety and reliability in autonomous systems. The research approach prioritized techniques that support computational efficiency, but these are often limited to non-critical tasks where minor inaccuracies are acceptable. Therefore, the frameworks applicability to non-accuracy tolerable, safety-critical scenarios remains constrained, as was evident in both the literature and the evaluation of the proposed approximation methods.

- **Balanced Accuracy and Precision:** Approximate computing is suitable for error tolerable tasks and computing environment, where a metrics (e.g., accuracy) can be traded or balanced against another (e.g., energy). For safety critical tasks where accuracy and precision are first priority, such computing practices can result to unexpected scenarios.
- **Deterministic vs Probabilistic:** The proposed approximation schemes can mostly be scoped within the probabilistic scenarios, and this nature makes it difficult to be adopted for safety-critical applications that expect a deterministic output for accurate functionality.
- **Edge and Real-time Processing:** While the proposed components for the energy-aware framework aims to target for memory, CPU and GPU requirements for the tested models. Dedicated in memory approximate computing strategies can be used for improving latency, throughput and memory footprint of these models which will also improve real-time inference.

• **Dataset and Case Studies Limitations:** These limitations are rooted in the methodology for selecting experimental testbeds and datasets, which followed systematic review and practical feasibility. The research approach involved analyzing widely adopted public datasets and open-source models to establish a generalizable foundation and ensure reproducibility. However, this focus on accessibility and benchmarked datasets means that some considerations, such as geographical variability, rare events, and real-world deployment complexity, are not fully captured. The inclusion and exclusion criteria established for the literature review and case selection inevitably shaped the scope of the results, making the findings representative but not exhaustive of all operational environments. These methodological decisions, while justified by current best practices, introduce known boundaries on the applicability and generalizability of the conclusions.

  - **Dataset Representativeness:** The datasets used may not fully represent the real-world variability of driving conditions across different geographical locations. This can limit the applicability of the findings to global contexts.
  - **Size and Scope:** The scope and size of the datasets might restrict the depth of training and evaluation, potentially overlooking certain edge cases or rare scenarios that could significantly impact model performance.
  - **Selection Bias:** The case studies chosen for testing and evaluating the framework were selected on the basis of open source datasets, popular classes in driving datasets and baseline models of the current generation. The results on datasets with alternative geographical or scenes may provide varied results and may not accurately reflect their effectiveness in non real-world conditions.

  - **Scalability to Real-World Applications:** While the dataset used for case studies provides insights into potential applications, they might not fully capture the complexities and scale of real-world deployment, particularly in highly dynamic environments with several wireless communication protocols.
  - **Underlying Assumptions:** The experiments are based on certain assumptions about dataset conditions, network stability, hardware uniformity, lab-based testbed and device behaviour, which might not hold true in all operational settings, potentially affecting the results.
  - **Simulation vs Real Deployment:** Many tests are conducted in simulated environments that, although complex, cannot perfectly resemble the unpredictability of real-world operations.

The research recommendations outlined here are addressing the limitations of this work. Each limitation whether in safety, dataset coverage, scalability, or assumptions about operational environments points to areas for further improvement. For example, challenges with safety-critical tasks and real-time processing motivate the exploration of hardware-software co-optimization and adaptive algorithms. The need for broader dataset coverage and real-world relevance supports more extensive cross-device testing and benchmarking. Differences between simulation and real deployment highlight the importance of predictive modeling and flexible, adaptive frameworks. By connecting these recommendations to specific limitations, this research provides a pathway for developing more robust, adaptable, and practical energy-aware edge AI systems for connected vehicles

Future Research Recommendation: With energy awareness as the primary target, this research and its contributions have complimented the subdomain of software-level approximation strategies and pinpointed promising directions for future research. To improve the operational efficiency of complex connected systems, here are potential research directions for further exploration.

- **Scaling the Framework Components and Cross-Device Testing and Adoption:**
  - A direction could be exploring strategies for scaling the Edge AI framework from a multidimensional perspective, which can involve enhancing adaptability, further improving power efficiency using application-specific tradeoffs (e.g., fault tolerance, memory footprints, latency etc.) in the conventional approximate computing space which currently looks from the higher level optimization of accuracy, performance and energy. The above exploration can be implemented using TinyML mechanisms. For scaling the framework, it is important to test and evaluate the performance from the application-specific tradeoffs on heterogeneous edge devices such as FPGAs and ASICs, where edge agentic models can be deployed, which based on deployment policies, select the best data processing locationswhether on-device, near-edge, or in the cloud, according to the real-time network conditions and energy considerations. The framework can dynamically adapt to the demands of different operational environments. This approach enables large-scale deployments to test the scalability and robustness of the framework across a network of

heterogeneous edge devices in urban locations, helping to identify operational variances and necessary adaptations for global applicability.

  - Further improvement in the framework can be developing a common API that integrate hardware-specific abstractions into the framework, enabling a common codebase to be executed efficiently across CPUs, GPUs, ASICs, and FPGAs. This includes creating adaptive algorithms that dynamically distribute workloads based on the computational characteristics and power profiles of each device type. Such a unified development approach would minimize dependency on specific hardware features and will improve the process of distribution of computational tasks. This strategy ensures that the framework remains flexible and scalable, supporting efficient cross-device functionality that is important for applications extending beyond vehicular systems and providing continuum into industrial IoT, smart cities, and healthcare.
  - An additional recommendation in the framework can also include implementation of benchmarking and profiling tools that evaluate performance and energy efficiency across varied application and hardware configurations. These tools would play an important role in optimizing resource allocation dynamically, ensuring that the system operates at peak efficiency without the hardware or device-specific constraints. By focusing on these areas, the Edge AI Framework can evolve into a more robust solution capable of supporting the diverse needs of modern connected systems, therefore enhancing its effectiveness and applicability across various domains and contributing to the broader goal of sustainable and efficient application deployment.

- **Stochastic Computing based HW-SW Co-optimization:**
  - With a focus on dynamic and context-aware approximation, a future research direction can be to explore the integration of stochastic computing principles into model approximation and partitioning strategies. This includes the formulation of a multi-parameter optimization problem, which should be addressed with the development of adaptive algorithms that depend on real-time dynamic operations, application prioritization or scheduler, and predicted environmental conditions (e.g., network conditions, bandwidth, computing resources manager, energy availability). Research could aim to use machine learning models to predict optimal stochastic bitstream lengths and computational fidelity (a measure of error resilience that can emulate the accurate output distribution) based on these dynamic contexts.
  - Another research scope is the design of a scalable, distributed architectural framework that enables the integration of stochastic computing-based model approximation within heterogeneous edge-cloud deployments. This framework would address the challenges of resource heterogeneity, network latency, and dynamic workload variations. The scope can be the creation of a dynamic resource orchestrator module that allocates and manages stochastic computing resources across vehicle computing units, edge nodes and cloud, driven by real-time application demands and environmental conditions. The framework should also include the exploration of efficient communication

protocols and data management strategies to minimize data transfer overhead and latency associated with stochastic bitstream processing. To ensure robust performance, an analysis of fault-tolerance mechanisms and reliability strategies is necessary, particularly in the case of hardware failures and network disruptions. Finally, the development of a comprehensive simulation and emulation environment will be important for thorough testing and validation of the framework under a wide range of operational conditions.

- **Predictive Modeling for Energy Saving:**
  - Future research work can also focus on comprehensive real-world tests to validate latency-tolerable services within vehicular networks, particularly emphasizing critical metrics such as response time and fault tolerance under various traffic scenarios and network conditions. Also, it is important to analyze the trade-offs between latency, energy consumption, and accuracy, especially in safety-critical applications. This involves a detailed examination of how variations in network latency and computational demands impact vehicle system reliability and response capabilities. Such studies are essential for developing methodologies that not only measure but also optimize energy usage and latency in connected vehicular environments. These methodologies should aim to dynamically adjust to real-time data processing demands and varying traffic conditions to enhance the balance between energy efficiency and operational latency.
  - Future research expanding and contributing to energy-aware edge AI systems can also look into advanced network optimization methods developed to address the dynamic challenges of vehicular networks. This research scope could include developing adaptive network routing protocols that dynamically adjust to the fluctuating conditions typical of vehicular movements and network heterogeneity within the urban infrastructure. These protocols aim to optimize data flow, minimizing latency and energy use, which is important for maintaining system efficiency and performance. Additionally, using predictive network management tools that utilize AI to estimate network loads and bottlenecks could be considered within next-generation automated solutions. By analyzing historical and real-time vehicular data, these tools would configure network settings to maintain optimal performance while balancing energy efficiency for the latency-tolerable and maybe also for safety-critical applications. This research direction can utilize AI and machine learning to improve cooperative vehicle operations and network-wide communication reliability with efficiency in connected and collaborative autonomous vehicular systems.

The research directions discussed above align with the growing interest among stakeholders in advancing the deployment of connected vehicles and their ecosystem. With this initiative, there is an increased focus on enhancing high-performance computing and data processing capabilities required by ML models responsible for performing varied tasks within the vehicle-edge-cloud ecosystem. The automotive community is also aiming to improve these technologies by addressing rising challenges such as an increase in energy consumption and reliance on centralized computing infrastructures. Multiple industry groups and sectors have initiated collaborations, forming consortia or working groups focused on edge AI solutions related to Automotive applications. These efforts aim to develop sustainable computing practices that will contribute to a greener future, ensuring that the evolution of connected vehicular systems aligns with broader societal goals for environmental sustainability. Such initiatives are in driving research that bridges technological advancements with eco-friendly practices, promoting a collective commitment in enhancing the efficiency and reliability of vehicular applications in a sustainable development manner.

A
List of Open-source Repositories

☞ Source code for framework components development and testing is available at: 4TU.

☞ Source code and instructions covered in Chapter 4 are available at: BiasDet.

☞ Approximation code and instructions covered in Chapter 5.1 are available at: AxC-Scheme 1.

☞ Code and instructions covered in Chapter 5.2 are available at: AxC-Scheme 2.

☞ Code and instructions covered in Chapter 5.3 are available at: AxC-Scheme 3.

☞ Code and instructions covered in Chapter 5.4 are available at: AxC-Scheme 4

☞ Model Partition and resource allocation code and instructions covered in Chapter 6 is available at ARASEC.

☞ Code and instructions covered in Chapter 7 are available at: AxCEnergy.

B
List of Acronyms

Acronym	Definition
5G	Fifth Generation Technology
ADAS	Advanced Driver Assistance Systems
AECC	Automotive Edge Computing Consortium
AxC	Approximate Computing
CAV	Connected Autonomous Vehicle
CNN	Convolutional Neural Network
Conv	Convolutional
CPU	Central Processing Unit
C-V2X	Cellular Vehicle-to-Everything
DNN	Deep Neural Network
DSRC	Dedicated Short Range Communication
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
FCW	Forward Collision Warning
FL	Federated Learning
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GPU	Graphical Processing Unit
HD Map	High-definition Map
IMU	Inertial Measurement Unit
ITS	Intelligent Transport Systems
LTE	Long Term Evolution
ML	Machine Learning
NX	Next Generation
OBU	On-board Unit
RNN	Recurrent Neural Network
ROS	Robot Operating System
RSU	Road Side Unit
SGD	Stochastic Gradient Descent
SLAM	Simultaneous Localization and Mapping
TPU	Tensor Processing Unit
UWB	Ultra Wideband
V2G	Vehicle-to-Grid
V2I	Vehicle-to-Infrastructure
V2N	Vehicle-to-Network
V2P	Vehicle-to-Pedestrian
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything
WiFi	Wireless Fidelity
WiMAX	Worldwide Interoperability for Microwave Access

Bibliography

[1] M. M. Rana and K. Hossain. ‘Connected and autonomous vehicles and infrastructures: A literature review’. In: *International Journal of Pavement Research and Technology* 16.2 (2023), pp. 264–284.

[2] O.-R. A. D. (committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Apr. 2021. DOI: https://doi.org/10.4271/J3016_202104. URL: https://doi.org/10.4271/J3016_202104.

[3] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang and W. Shi. ‘Computing systems for autonomous driving: State of the art and challenges’. In: *IEEE Internet of Things Journal* 8.8 (2020), pp. 6469–6486.

[4] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii and T. Azumi. ‘Autoware on board: Enabling autonomous vehicles with embedded systems’. In: *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE. 2018, pp. 287–296.

[5] S. Hakak, T. R. Gadekallu, P. K. R. Maddikunta, S. P. Ramu, M. Parimala, C. De Alwis and M. Liyanage. ‘Autonomous Vehicles in 5G and beyond: A Survey’. In: *Vehicular Communications* 39 (2023), p. 100551.

[6] W. Shi and L. Liu. ‘Smart Infrastructure for Autonomous Driving’. In: *Computing Systems for Autonomous Driving*. Springer, 2021, pp. 173–190.

[7] P. Arthurs, L. Gillam, P. Krause, N. Wang, K. Halder and A. Mouzakitis. ‘A Taxonomy and Survey of Edge Cloud Computing for Intelligent Transportation Systems and Connected Vehicles’. In: *IEEE Transactions on Intelligent Transportation Systems* (2021), pp. 1–16. DOI: 10.1109/TITS.2021.3084396.

[8] R. Mur-Artal, J. M. M. Montiel and J. D. Tardos. ‘ORB-SLAM: a versatile and accurate monocular SLAM system’. In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.

[9] J. Czarnowski, T. Laidlow, R. Clark and A. J. Davison. ‘Deepfactors: Real-time probabilistic dense monocular slam’. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 721–728.

[10] S. Kochanthara, T. Singh, A. Forrai and L. Cleophas. ‘Safety of Perception Systems for Automated Driving: A Case Study on Apollo’. In: *ACM Trans. Softw. Eng. Methodol.* 33.3 (2024). DOI: 10.1145/3631969.

[11] L. Di Lillo, T. Gode, X. Zhou, M. Atzei, R. Chen and T. Victor. ‘Comparative safety performance of autonomous-and human drivers: A real-world case study of the Waymo Driver’. In: *Heliyon* 10.14 (2024).

[12] K. Wong, Y. Gu and S. Kamijo. 'Mapping for autonomous driving: Opportunities and challenges'. In: *IEEE Intelligent Transportation Systems Magazine* 13.1 (2020), pp. 91–106. DOI: 10.1109/MITS.2020.3014152.

[13] X. Deng, L. Wang, J. Gui, P. Jiang, X. Chen, F. Zeng and S. Wan. 'A review of 6G autonomous intelligent transportation systems: Mechanisms, applications and challenges'. In: *Journal of Systems Architecture* 142 (2023), p. 102929.

[14] A. Soltoggio, E. Ben-Iwhiwhu, V. Braverman, E. Eaton, B. Epstein, Y. Ge, L. Halperin, J. How, L. Itti, M. A. Jacobs *et al.* 'A collective AI via lifelong learning and sharing at the edge'. In: *Nature Machine Intelligence* 6.3 (2024), pp. 251–264.

[15] F. Bouali, J. Pinola, V. Karyotis, B. Wissingh, M. Mitrou, P. Krishnan and K. Moessner. '5G for Vehicular Use Cases: Analysis of Technical Requirements, Value Propositions and Outlook'. In: *IEEE Open Journal of Intelligent Transportation Systems* 2 (2021), pp. 73–96. DOI: 10.1109/OJITS.2021.3072220.

[16] M. Elassy, M. Al-Hattab, M. Takruri and S. Badawi. 'Intelligent transportation systems for sustainable smart cities'. In: *Transportation Engineering* (2024), p. 100252.

[17] X. Han, Z. Meng, X. Xia, X. Liao, B. Y. He, Z. Zheng, Y. Wang, H. Xiang, Z. Zhou, L. Gao *et al.* 'Foundation intelligence for smart infrastructure services in transportation 5.0'. In: *IEEE Transactions on Intelligent Vehicles* 9.1 (2024), pp. 39–47.

[18] AECC. *Proofs of Concept - Automotive Edge Computing Consortium — aecc.org*. https://aecc.org/proof-of-concepts/entry/956/. [Accessed 06-12-2024]. 2024.

[19] S. Barua. 'Flood of data will get generated in autonomous cars'. In: *Auto Tech Review, nd [https://autotechreview. com/features/flood-of-data-will-get-generated-in-autonomouscars]* ().

[20] D. Katare, D. Perino, J. Nurmi, M. Warnier, M. Janssen and A. Y. Ding. 'A survey on approximate edge ai for energy efficient autonomous driving services'. In: *IEEE Communications Surveys & Tutorials* (2023). DOI: 10.1109/COMST.2023.3302474.

[21] X. Chen, Y. Deng, H. Ding, G. Qu, H. Zhang, P. Li and Y. Fang. 'Vehicle as a service (vaas): Leverage vehicles to build service networks and capabilities for smart cities'. In: *arXiv preprint arXiv:2304.11397* (2023).

[22] S. Sudhakar, V. Sze and S. Karaman. 'Data Centers on Wheels: Emissions From Computing Onboard Autonomous Vehicles'. In: *IEEE Micro* (2023), pp. 29–39.

[23] K. Rajashekara. 'Energy impacts of autonomous vehicles-Present and the future'. In: *iEnergy* 3.2 (2024), pp. 73–74. DOI: 10.23919/IEN.2024.0012.

[24] M. Krail, J. Hellekes, U. Schneider, E. Dütschke, M. Schellert, D. Rüdiger, A. Steindl, I. Luchmann, V. WaSSmuth, H. Flämig *et al.* 'Energie-und Treibhausgaswirkungen des automatisierten und vernetzten Fahrens im StraSSenverkehr'. In: *Final report of the study on behalf of the German Federal Ministry of Transport and Digital Infrastructure. Karlsruhe, Germany* (2019).

[25] K. Antonakoglou, N. Brahmi, T. Abbas, A. E. Fernandez Barciela, M. Boban, K. Cordes, M. Fallgren, L. Gallo, A. Kousaridas, Z. Li *et al.* 'On the needs and requirements arising from connected and automated driving'. In: *Journal of Sensor and Actuator Networks* (2020), p. 24.

[26] B. Minto. *The pyramid principle: logic in writing and thinking.* Pearson Education, 2009.

[27] H. J. Damsgaard, A. Grenier, D. Katare, Z. Taufique, S. Shakibhamedan, T. Troccoli, G. Chatzitsompanis, A. Kanduri, A. Ometov, A. Y. Ding *et al.* 'Adaptive approximate computing in edge AI and IoT applications: A review'. In: *Journal of Systems Architecture* (2024), p. 103114. DOI: https://doi.org/10.1016/j.sysarc.2024.103114.

[28] D. S. Hochba. 'Approximation Algorithms for NP-Hard Problems'. In: *SIGACT News* 28.2 (June 1997), pp. 40–52. DOI: 10.1145/261342.571216.

[29] J. Skinner. *Principles of Approximate Computations.* H. Holt, 1876. URL: https://books.google.nl/books?id=CuM2AAAAMAAJ.

[30] H. Sun, A. Abdelwahab and B. Onaral. 'Linear approximation of transfer function with a pole of fractional power'. In: *IEEE Transactions on Automatic Control* 29.5 (1984), pp. 441–444. DOI: 10.1109/TAC.1984.1103551.

[31] J. Han and M. Orshansky. 'Approximate computing: An emerging paradigm for energy-efficient design'. In: *2013 18th IEEE European Test Symposium (ETS).* 2013, pp. 1–6. DOI: 10.1109/ETS.2013.6569370.

[32] L. Sekanina. 'Introduction to approximate computing: Embedded tutorial'. In: *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS).* 2016, pp. 1–6. DOI: 10.1109/DDECS.2016.7482460.

[33] S. Venkataramani, S. T. Chakradhar, K. Roy and A. Raghunathan. 'Approximate computing and the quest for computing efficiency'. In: *Proceedings of the 52nd Annual Design Automation Conference.* Association for Computing Machinery, 2015. ISBN: 9781450335201.

[34] T.-J. Yang, Y.-H. Chen, J. Emer and V. Sze. 'A method to estimate the energy consumption of deep neural networks'. In: *2017 51st asilomar conference on signals, systems, and computers.* IEEE. 2017, pp. 1916–1920.

[35] A. Mohan, S. Sripad, P. Vaishnav and V. Viswanathan. 'Trade-offs between automation and light vehicle electrification'. In: *Nature Energy* 5.7 (2020), pp. 543–549.

[36] J. Henkel, H. Li, A. Raghunathan, M. B. Tahoori, S. Venkataramani, X. Yang and G. Zervakis. 'Approximate computing and the efficient machine learning expedition'. In: *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design.* 2022, pp. 1–9.

[37] K. V. Krishnan, A. Satish and P. raj Krishnan. 'Design of energy efficient approximate subtractors and restoring dividers for error tolerant applications'. In: *Microelectronics Journal* 131 (2023).

[38] S. K. Ghosh, A. Raha and V. Raghunathan. 'Energy-efficient approximate edge inference systems'. In: *ACM Transactions on Embedded Computing Systems* 22.4 (2023), pp. 1–50.

[39] D. Katare, S. Shakibhamedan, N. Amirafshar, N. Taherinejad, A. Jantsch, M. Janssen and A. Y. Ding. 'Approximation Strategies for Vision Models on Edge Devices: An Accuracy-Efficiency Trade-off'. In: *TechRxiv-Preprint* (2024).

[40] D. Katare and A. Y. Ding. 'Energy-efficient edge approximation for connected vehicular services'. In: *2023 57th Annual Conference on Information Sciences and Systems (CISS)*. IEEE. 2023, pp. 1–6. DOI: 10.1109/CISS56502.2023.10089724.

[41] M. Fabjani, O. Machidon, H. Sharif, Y. Zhao, S. Misailovi and V. Pejovi. 'Mobiprox: Supporting Dynamic Approximate Computing on Mobiles'. In: *IEEE Internet of Things Journal* 11.9 (2024), pp. 16873–16886. DOI: 10.1109/JIOT.2024.3365957.

[42] S. Huo, J. S. Morris and H. Zhu. 'Ultra-Fast Approximate Inference Using Variational Functional Mixed Models'. In: *Journal of Computational and Graphical Statistics* 32.2 (2023), pp. 353–365.

[43] D. Katare, S. Leroux, M. Janssen and A. Y. Ding. 'Approximating vision transformers for edge: variational inference and mixed-precision for multi-modal data'. In: *Computing* 107.3 (2025), p. 71. DOI: 10.1007/s00607-025-01427-w.

[44] T. P. Minka. 'A family of algorithms for approximate Bayesian inference'. PhD thesis. Massachusetts Institute of Technology, 2001.

[45] T. P. Minka. 'A comparison of numerical optimizers for logistic regression'. In: *Citeseer* (2003).

[46] D. Katare and A. Y. Ding. 'Energy-efficient Edge Approximation for Connected Vehicular Services'. In: *2023 57th Annual Conference on Information Sciences and Systems (CISS)*. 2023, pp. 1–6. DOI: 10.1109/CISS56502.2023.10089724.

[47] S. Küfeolu. 'SDG-9: industry, innovation and infrastructure'. In: *Emerging technologies: value creation for sustainable development*. Springer, 2022, pp. 349–369.

[48] S. Küfeolu. 'SDG-11: Sustainable cities and communities'. In: *Emerging technologies: Value creation for sustainable development*. Springer, 2022, pp. 385–408.

[49] M. G. Augusto, J. B. Krug, B. Acar, F. Sivrikaya and S. Albayrak. 'Debunking the Myth of High Consumption: Power Realities in Autonomous Vehicles*'. In: *2024 IEEE Intelligent Vehicles Symposium (IV)*. 2024, pp. 2869–2875. DOI: 10.1109/IV55156.2024.10588430.

[50] S. Feng, X. Yan, H. Sun, Y. Feng and H. X. Liu. 'Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment'. In: *Nature communications* 12.1 (2021), p. 748.

[51] M. Lechner, R. Hasani, A. Amini, T. A. Henzinger, D. Rus and R. Grosu. 'Neural circuit policies enabling auditable autonomy'. In: *Nature Machine Intelligence* 2.10 (2020), pp. 642–652.

[52] N. Alizadeh and F. Castor. 'Green ai: A preliminary empirical study on energy consumption in dl models across different runtime infrastructures'. In: *Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI*. 2024, pp. 134–139.

[53] B. Buko, M. Michálek, K. Papierniková and K. Zábovská. 'Smart Mobility and Aspects of Vehicle-to-Infrastructure: A Data Viewpoint'. In: *Applied Sciences* 11.22 (2021), p. 10514.

[54] Z. Yu, J. Hu, G. Min, Z. Zhao, W. Miao and M. S. Hossain. 'Mobility-aware proactive edge caching for connected vehicles using federated learning'. In: *IEEE Transactions on Intelligent Transportation Systems* 22.8 (2020), pp. 5341–5351.

[55] Y. Han, H. Zhang, H. Li, Y. Jin, C. Lang and Y. Li. 'Collaborative perception in autonomous driving: Methods, datasets and challenges'. In: *arXiv preprint arXiv:2301.06262* (2023).

[56] D. Katare and M. El-Sharkawy. 'Embedded system enabled vehicle collision detection: an ANN classifier'. In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2019, pp. 0284–0289. DOI: 10.1109/CCWC.2019.8666562.

[57] R. Wang, L. Liu and W. Shi. 'HydraSpace: computational data storage for autonomous vehicles'. In: *2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC)*. IEEE. 2020, pp. 70–77.

[58] D. Katare, M. Zhou, Y. Chen, M. Janssen and A. Y. Ding. 'Energy-Aware Vision Model Partitioning for Edge AI'. In: *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*. SAC '25. 2025, pp. 1–8. DOI: 10.1145/3672608.3707792.

[59] W. Gao and X. Yang. 'A Joint Resource Allocation and Task Offloading Scheme for Energy-aware and Latency Constrained Vehicular Edge Computing Network'. In: *2024 IEEE/CIC International Conference on Communications in China (ICCC)*. IEEE. 2024, pp. 167–172.

[60] F. Golbabaei, A. Paz, T. Yigitcanlar and J. Bunker. 'Navigating autonomous demand responsive transport: stakeholder perspectives on deployment and adoption challenges'. In: *International Journal of Digital Earth* 17.1 (2024), p. 2297848.

[61] *ApolloAuto: An open autonomous driving platform*. ApolloAuto. URL: https://apollo.auto/developer.html.

[62] E. Ackerman. 'What Full Autonomy Means for the Waymo Driver'. In: *IEEE Spectrum, Mar* (2021).

[63] Y. Cao, S. Derrible, M. Le Pira and H. Du. 'Advanced transport systems: the future is sustainable and technology-enabled'. In: *Scientific reports* 14.1 (2024), p. 9429.

[64] K. Peffers, M. Rothenberger and B. Kuechler. *Design Science Research in Information Systems. Advances in Theory and Practice: 7th International Conference, DESRIST 2012, Las Vegas, NV, USA, May 14-15, 2012. Proceedings*. Vol. 7286. Jan. 2012. ISBN: 978-3-642-29862-2. DOI: 10.1007/978-3-642-29863-9.

[65] A. R. Hevner. ‘A three cycle view of design science research’. In: *Scandinavian journal of information systems* 19.2 (2007), p. 4.

[66] P. Offermann, S. Blom, M. Schönherr and U. Bub. ‘Artifact types in information systems design science–a literature review’. In: *International Conference on Design Science Research in Information Systems*. Springer. 2010, pp. 77–92.

[67] S. T. March and G. F. Smith. ‘Design and natural science research on information technology’. In: *Decision support systems* 15.4 (1995), pp. 251–266.

[68] M. Turner, B. Kitchenham, P. Brereton, S. Charters and D. Budgen. ‘Does the technology acceptance model predict actual use? A systematic literature review’. In: *Information and software technology* 52.5 (2010), pp. 463–479.

[69] R. K. Yin. *Case study research: Design and methods*. Vol. 5. sage, 2009.

[70] J. Tang, S. Liu, L. Liu, B. Yu and W. Shi. ‘LoPECS: A Low-Power Edge Computing System for Real-Time Autonomous Driving Services’. In: *IEEE Access* 8 (2020), pp. 30467–30479.

[71] Y. Wang, S. Liu, X. Wu and W. Shi. ‘CAVBench: A benchmark suite for connected and autonomous vehicles’. In: *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE. 2018, pp. 30–42.

[72] X. Zhang, Y. Wang, S. Lu, L. Liu, W. Shi *et al.* ‘OpenEI: An open framework for edge intelligence’. In: *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2019, pp. 1840–1851.

[73] J. Tang, S. Liu, B. Yu and W. Shi. ‘PI-Edge: A Low-Power Edge Computing System for Real-Time Autonomous Driving Services’. In: *CoRR* abs/1901.04978 (2019).

[74] J. Tang, S. Liu, L. Liu, B. Yu and W. Shi. ‘LoPECS: A low-power edge computing system for real-time autonomous driving services’. In: *IEEE Access* 8 (2020), pp. 30467–30479.

[75] D. Katare and M. El-Sharkawy. ‘Autonomous Embedded System Enabled 3-D Object Detector:(with Point Cloud and Camera)’. In: *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE. 2019, pp. 1–6. DOI: 10.1109/ICVES.2019.8906442.

[76] N. Amirafshar, A. S. Baroughi, H. S. Shahhoseini and N. TaheriNejad. ‘An Approximate Carry Disregard Multiplier with Improved Mean Relative Error Distance and Probability of Correctness’. In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. 2022, pp. 46–52.

[77] X. Yang, S. Chaudhuri, L. Naviner and L. Likforman. ‘Quad-Approx CNNs for Embedded Object Detection Systems’. In: *2020 27th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 2020, pp. 1–4.

[78] V. Kumar and R. Kant. ‘Approximate computing for machine learning’. In: *Proceedings of 2nd International Conference on Communication, Computing and Networking: ICCCN 2018, NITTTR Chandigarh, India*. Springer. 2019, pp. 607–613.

[79] S. Rajput, T. Widmayer, Z. Shang, M. Kechagia, F. Sarro and T. Sharma. 'Enhancing Energy-Awareness in Deep Learning through Fine-Grained Energy Measurement'. In: *ACM Transactions on Software Engineering and Methodology* 33.8 (2024), pp. 1–34.

[80] A. Look. 'Deterministic Approximations for Deep State-Space Models'. In: (2023).

[81] M. Althoff and A. Mergel. 'Comparison of Markov chain abstraction and Monte Carlo simulation for the safety assessment of autonomous cars'. In: *IEEE Transactions on Intelligent Transportation Systems* 12.4 (2011), pp. 1237–1247.

[82] Y. Wu, C. Chen, W. Xiao, X. Wang, C. Wen, J. Han, X. Yin, W. Qian and C. Zhuo. 'A Survey on Approximate Multiplier Designs for Energy Efficiency: From Algorithms to Circuits'. In: *arXiv preprint arXiv:2301.12181* (2023).

[83] C. Guo, L. Zhang, X. Zhou, W. Qian and C. Zhuo. 'A Reconfigurable Approximate Multiplier for Quantized CNN Applications'. In: *25th Asia and South Pacific Design Automation Conference, ASP-DAC 2020, Beijing, China, January 13-16, 2020.* IEEE, 2020, pp. 235–240.

[84] C. D. la Parra, A. Guntoro and A. Kumar. 'ProxSim: GPU-based Simulation Framework for Cross-Layer Approximate DNN Optimization'. In: *2020 Design, Automation & Test in Europe Conference & Exhibition, DATE 2020, Grenoble, France, March 9-13, 2020.* IEEE, 2020, pp. 1193–1198.

[85] H. Tann, S. Hashemi and S. Reda. 'Lightweight Deep Neural Network Accelerators Using Approximate SW/HW Techniques'. In: *Approximate Circuits, Methodologies and CAD.* Springer, 2019, pp. 289–305.

[86] J. Tee and D. P. Taylor. 'A quantized representation of probability in the brain'. In: *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* 5.1 (2019), pp. 19–29.

[87] J. Feng, Z. Liu, C. Wu and Y. Ji. 'AVE: Autonomous Vehicular Edge Computing Framework with ACO-Based Scheduling'. In: *IEEE Trans. Veh. Technol.* 66.12 (2017), pp. 10660–10675.

[88] H. Ibn-Khedher, M. Laroui, M. B. Mabrouk, H. Moungla, H. Afifi, A. N. Oleari and A. E. Kamal. 'Edge Computing Assisted Autonomous Driving Using Artificial Intelligence'. In: *2021 International Wireless Communications and Mobile Computing (IWCMC).* IEEE. 2021, pp. 254–259.

[89] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He and K. Chan. 'When edge meets learning: Adaptive control for resource-constrained distributed machine learning'. In: *IEEE INFOCOM 2018-IEEE conference on computer communications.* IEEE. 2018, pp. 63–71.

[90] C. Li, Y. Zhang and Y. Luo. 'A federated learning-based edge caching approach for mobile edge computing-enabled intelligent connected vehicles'. In: *IEEE Transactions on Intelligent Transportation Systems* 24.3 (2022), pp. 3360–3369.

[91] C. Liang, Y. Zhao, Z. Gao, K. Cheng, B. Wang and L. Huang. 'EALSO: joint energy-aware and latency-sensitive task offloading for artificial Intelligence of Things in vehicular fog computing'. In: *Wireless Networks* (2024), pp. 1–17.

[92] C. Lin, C. Li and J. Liu. ‘A QoS-Aware Training Framework for ViT Compression, Partition, and Distillation’. In: *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*. IEEE. 2024, pp. 1–2.

[93] H. Liang, Q. Sang, C. Hu, D. Cheng, X. Zhou, D. Wang, W. Bao and Y. Wang. ‘DNN surgery: Accelerating DNN inference on the edge through layer partitioning’. In: *IEEE transactions on Cloud Computing* 11.3 (2023), pp. 3111–3125.

[94] A. Ravi, V. Chaturvedi and M. Shafique. ‘ViT4Mal: Lightweight Vision Transformer for Malware Detection on Edge Devices’. In: *ACM Transactions on Embedded Computing Systems* 22.5s (2023), pp. 1–26.

[95] Y.-C. Wang, J. Xue, C. Wei and C.-C. J. Kuo. ‘An overview on generative AI at scale with Edge-Cloud Computing’. In: *IEEE Open Journal of the Communications Society* (2023).

[96] A. N. Houssam Kanso and E. Exposito. ‘A Review of Energy Aware Cyber-Physical Systems’. In: *Cyber-Physical Systems* 10.1 (2024), pp. 1–42.

[97] T. Bahreini, M. Brocanelli and D. Grosu. ‘VECMAN: A framework for energy-aware resource management in vehicular edge computing systems’. In: *IEEE Transactions on Mobile Computing* (2021).

[98] X. Kong, G. Duan, M. Hou, G. Shen, H. Wang, X. Yan and M. Collotta. ‘Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles’. In: *IEEE Transactions on Industrial Informatics* 18.9 (2022), pp. 6308–6316.

[99] Z. Song, M. Xie, J. Luo, T. Gong and W. Chen. ‘A Carbon-Aware Framework for Energy-Efficient Data Acquisition and Task Offloading in Sustainable AIoT Ecosystems’. In: *IEEE Internet of Things Journal* (2024).

[100] J. Wang, C. Lan, C. Liu, Y. Ouyang, T. Qin, W. Lu, Y. Chen, W. Zeng and P. Yu. ‘Generalizing to unseen domains: A survey on domain generalization’. In: *IEEE Transactions on Knowledge and Data Engineering* (2022).

[101] M. Hardt, E. Price and N. Srebro. ‘Equality of opportunity in supervised learning’. In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2016, pp. 3323–3331.

[102] D. Danks and A. J. London. ‘Algorithmic Bias in Autonomous Systems.’ In: *IJCAI*. Vol. 17. 2017, pp. 4691–4697.

[103] D. Katare and M. El-Sharkawy. ‘Collision warning system: embedded enabled (RTMaps with NXP BLBX2)’. In: *2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE. 2018, pp. 1–6. DOI: 10.1109/ISSPIT.2018.8705101.

[104] S. Zhao, H. Ren, A. Yuan, J. Song, N. Goodman and S. Ermon. ‘Bias and generalization in deep generative models: An empirical study’. In: *Advances in Neural Information Processing Systems* 31 (2018).

[105] I. Rafegas, M. Vanrell, L. A. Alexandre and G. Arias. ‘Understanding trained CNNs by indexing neuron selectivity’. In: *Pattern Recognition Letters* 136 (2020), pp. 318–325.

[106] A. Vobecky, M. Uricar, D. Hurych and R. Skoviera. 'Advanced Pedestrian Dataset Augmentation for Autonomous Driving'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*. Oct. 2019.

[107] S. Madan, T. Henry, J. Dozier, H. Ho, N. Bhandari, T. Sasaki, F. Durand, H. Pfister and X. Boix. 'Biased-Cars Dataset'. In: *arXiv preprint arXiv:2301.00493* (2021).

[108] Y. Chen and J. Joo. 'Understanding and Mitigating Annotation Bias in Facial Expression Recognition'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 14980–14991.

[109] H. Jiang and O. Nachum. 'Identifying and correcting label bias in machine learning'. In: *International conference on artificial intelligence and statistics*. PMLR. 2020, pp. 702–712.

[110] J. P. Robinson, G. Livitz, Y. Henon, C. Qin, Y. Fu and S. Timoner. 'Face recognition: too bias, or not too bias?' In: *Proceedings of the ieee/cvf conference on computer vision and pattern recognition workshops*. 2020, pp. 0–1.

[111] Z. Cui, Y. Zhang and Q. Ji. 'Label error correction and generation through label relationships'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 2020, pp. 3693–3700.

[112] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Beijbom. 'nuscenes: A multimodal dataset for autonomous driving'. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 11621–11631.

[113] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen and D. Anguelov. 'Scalability in Perception for Autonomous Driving: Waymo Open Dataset'. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 2446–2454.

[114] J.-W. Hong, I. Cruz and D. Williams. 'AI, you can drive my car: How we evaluate human drivers vs. self-driving cars'. In: *Computers in Human Behavior* 125 (2021), p. 106944.

[115] D. Katare, N. Kourtellis, S. Park, D. Perino, M. Janssen and A. Y. Ding. 'Bias detection and generalization in AI algorithms on edge for autonomous driving'. In: *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. IEEE. 2022, pp. 342–348. DOI: `10.1109/SEC54971.2022.00050`.

[116] S. Fabbrizzi, S. Papadopoulos, E. Ntoutsi and I. Kompatsiaris. 'A survey on bias in visual datasets'. In: *Computer Vision and Image Understanding* 223 (2022), p. 103552. DOI: `10.1016/j.cviu.2022.103552`.

[117] B. Wilson, J. Hoffman and J. Morgenstern. 'Predictive inequity in object detection'. In: *arXiv preprint arXiv:1902.11097* (2019).

[118] D. Lee and J. Kim. ‘Resolving class imbalance for lidar-based object detector by dynamic weight average and contextual ground truth sampling’. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2023, pp. 682–691. DOI: 10.1109/WACV56688.2023.00075.

[119] J. Siegel and G. Pappas. ‘Morals, ethics, and the technology capabilities and limitations of automated and self-driving vehicles’. In: *AI & SOCIETY* (2021), pp. 1–14. DOI: 10.1007/s00146-021-01277-y.

[120] M. Saini and S. Susan. ‘Bag-of-Visual-Words codebook generation using deep features for effective classification of imbalanced multi-class image datasets’. In: *Multimedia Tools and Applications* 80 (2021), pp. 20821–20847.

[121] J. Buolamwini and T. Gebru. ‘Gender shades: Intersectional accuracy disparities in commercial gender classification’. In: *Conference on fairness, accountability and transparency*. PMLR. 2018, pp. 77–91.

[122] M. Mazumder, C. Banbury, X. Yao, B. Karla, W. Gaviria Rojas, S. Diamos, G. Diamos, L. He, A. Parrish, H. R. Kirk *et al.* ‘Dataperf: Benchmarks for data-centric ai development’. In: *Advances in Neural Information Processing Systems* 36 (2024).

[123] H. Alibrahim and S. A. Ludwig. ‘Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization’. In: *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE. 2021, pp. 1551–1559.

[124] A. Wang, A. Liu, R. Zhang, A. Kleiman, L. Kim, D. Zhao, I. Shirai, A. Narayanan and O. Russakovsky. ‘REVISE: A tool for measuring and mitigating bias in visual datasets’. In: *International Journal of Computer Vision* 130.7 (2022), pp. 1790–1810.

[125] N. A. Stanton, P. M. Salmon, G. H. Walker and M. Stanton. ‘Models and methods for collision analysis: A comparison study based on the Uber collision with a pedestrian’. In: *Safety Science* 120 (2019), pp. 117–128.

[126] Y. Nie, A. S. Zamzam and A. Brandt. ‘Resampling and data augmentation for short-term PV output prediction based on an imbalanced sky images dataset using convolutional neural networks’. In: *Solar Energy* 224 (2021), pp. 341–354.

[127] Z. S. Rubaidi, B. B. Ammar and M. B. Aouicha. ‘Fraud detection using large-scale imbalance dataset’. In: *International Journal on Artificial Intelligence Tools* 31.08 (2022), p. 2250037.

[128] R. Li, Y. Wang, F. Liang, H. Qin, J. Yan and R. Fan. ‘Fully quantized network for object detection’. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 2810–2819.

[129] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer. ‘SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size’. In: *arXiv preprint arXiv:1602.07360* (2016).

[130] T. Yin, X. Zhou and P. Krahenbuhl. ‘Center-based 3d object detection and tracking’. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 11784–11793.

[131] L. Fan, F. Wang, N. Wang and Z.-X. Zhang. ‘Fully sparse 3d object detection’. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 351–363.

[132] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.* 'An image is worth 16x16 words: Transformers for image recognition at scale'. In: *arXiv preprint arXiv:2010.11929* (2020).

[133] A. Ali, T. Schnake, O. Eberle, G. Montavon, K.-R. Müller and L. Wolf. 'XAI for transformers: Better explanations through conservative propagation'. In: *International Conference on Machine Learning*. PMLR. 2022, pp. 435–451.

[134] G. Montavon, A. Binder, S. Lapuschkin, W. Samek and K.-R. Müller. 'Layer-wise relevance propagation: an overview'. In: *Explainable AI: interpreting, explaining and visualizing deep learning* (2019), pp. 193–209.

[135] T.-Y. Ross and G. Dollár. 'Focal loss for dense object detection'. In: *proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2980–2988.

[136] R. Keshari, M. Vatsa, R. Singh and A. Noore. 'Learning structure and strength of CNN filters for small sample size training'. In: *proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9349–9358.

[137] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes *et al.* 'Argoverse 2: Next generation datasets for self-driving perception and forecasting'. In: *arXiv preprint arXiv:2301.00493* (2023).

[138] J. Mao, H. Yang, A. Li, H. Li and Y. Chen. 'TPrune: Efficient Transformer Pruning for Mobile Devices'. In: *ACM Trans. Cyber Phys. Syst.* 5.3 (2021), 26:1–26:22.

[139] C. Yu, J. Wang, C. Peng, C. Gao, G. Yu and N. Sang. 'Bisenet: Bilateral segmentation network for real-time semantic segmentation'. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 325–341.

[140] G. Habib and S. Qureshi. 'Optimization and acceleration of convolutional neural networks: A survey'. In: *Journal of King Saud University - Computer and Information Sciences* 34 (2022), pp. 4244–4268.

[141] R. Pilipovi, V. Risojevi, J. Boi, P. Buli and U. Lotri. 'An approximate GEMM unit for energy-efficient object detection'. In: *Sensors* 21.12 (2021), p. 4195.

[142] S. Shakibhamedan, A. Aminifar, N. Taherinejad and A. Jantsch. 'Ease: Energy optimization through adaptation–a review of runtime energy-aware approximate deep learning algorithms'. In: *Authorea Preprints* (2024).

[143] N. TaheriNejad and S. Shakibhamedan. 'Energy-aware Adaptive Approximate Computing for Deep Learning Applications'. In: *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2022, pp. 328–328.

[144] E. Jagadeeswara Rao and P. Samundiswary. 'A review of approximate multipliers and its applications'. In: *Advances in Automation, Signal Processing, Instrumentation, and Control: Select Proceedings of i-CASIC 2020* (2021), pp. 1381–1392.

[145] C. Tang, K. Ouyang, Z. Wang, Y. Zhu, W. Ji, Y. Wang and W. Zhu. 'Mixed-Precision Neural Network Quantization via Learned Layer-Wise Importance'. In: *European Conference on Computer Vision*. Springer. 2022, pp. 259–275.

[146] P. Xu, X. Zhu and D. A. Clifton. 'Multimodal Learning With Transformers: A Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.10 (2023), pp. 12113–12132.

[147] S. Shakibhamedan, A. Jahanjoo, A. Aminifar, N. Amirafshar, N. TaheriNejad and A. Jantsch. 'An Analytical Approach to Enhancing DNN Efficiency and Accuracy Using Approximate Multiplication'. In: *2nd Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization (WANT@ICML 2024)*. 2024.

[148] S. Shakibhamedan, N. Amirafshar, A. S. Baroughi, H. S. Shahhoseini and N. TaheriNejad. 'ACE-CNN: Approximate Carry Disregard Multipliers for Energy-Efficient CNN-Based Image Classification'. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 71.5 (2024), pp. 2280–2293.

[149] K. Sun and F. Nielsen. 'A Geometric Modeling of Occam's Razor in Deep Learning'. In: *arXiv preprint arXiv:1905.11027* (2019).

[150] H. He, J. Cai, J. Liu, Z. Pan, J. Zhang, D. Tao and B. Zhuang. 'Pruning self-attentions into convolutional layers in single path'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2024).

[151] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu *et al.* 'A survey on vision transformer'. In: *IEEE transactions on pattern analysis and machine intelligence* 45.1 (2022), pp. 87–110.

[152] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh and H. Wu. 'Mixed Precision Training'. In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=r1gs9JgRZ.

[153] Y. S. Tai and A.-Y. Wu. 'MPTQ-ViT: Mixed-Precision Post-Training Quantization for Vision Transformer'. In: *ArXiv* abs/2401.14895 (2024).

[154] P. Yin, J. Lyu, S. Zhang, S. Osher, Y. Qi and J. Xin. 'Understanding straight-through estimator in training activation quantized neural nets'. In: *International Conference on Learning Representations*. 2019.

[155] A. Ando, S. Gidaris, A. Bursuc, G. Puy, A. Boulch and R. Marlet. 'RangeViT: Towards Vision Transformers for 3D Semantic Segmentation in Autonomous Driving'. In: *CVPR*. 2023.

[156] R. Q. Hu *et al.* 'Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning'. In: *IEEE Transactions on Vehicular Technology* 67.11 (2018), pp. 10190–10203.

[157] X. Xu, X. Zhou, X. Zhou, M. Bilal, L. Qi, X. Xia and W. Dou. 'Distributed edge caching for zero trust-enabled connected and automated vehicles: A multi-agent reinforcement learning approach'. In: *IEEE Wireless Communications* 31.2 (2024), pp. 36–41.

[158] Z. Ning, K. Zhang, X. Wang, L. Guo, X. Hu, J. Huang, B. Hu and R. Y. Kwok. 'Intelligent edge computing in internet of vehicles: A joint computation offloading and caching solution'. In: *IEEE Transactions on Intelligent Transportation Systems* 22.4 (2020), pp. 2212–2225.

[159] D. Katare, E. Marin, N. Kourtellis, M. Janssen and A. Y. Ding. 'ARASEC: Adaptive Resource Allocation and Model Training for Serverless EdgeCloud Computing'. In: *IEEE Internet Computing* 28.6 (2024), pp. 17–27. DOI: 10.1109/MIC.2024.3514670.

[160] H. Wu, W. J. Knottenbelt and K. Wolter. 'An efficient application partitioning algorithm in mobile environments'. In: *IEEE Transactions on Parallel and Distributed Systems* 30.7 (2019), pp. 1464–1480.

[161] W. Zhang, N. Wang, L. Li and T. Wei. 'Joint compressing and partitioning of CNNs for fast edge-cloud collaborative intelligence for IoT'. In: *Journal of Systems Architecture* 125 (2022), p. 102461.

[162] R. Xin, D. Jakoveti and U. A. Khan. 'Distributed Nesterov Gradient Methods Over Arbitrary Graphs'. In: *IEEE Signal Processing Letters* 26.8 (2019), pp. 1247–1251. DOI: 10.1109/LSP.2019.2925537.

[163] X. Lian, W. Zhang, C. Zhang and J. Liu. 'Asynchronous decentralized parallel stochastic gradient descent'. In: *International Conference on Machine Learning*. PMLR. 2018, pp. 3043–3052.

[164] H. Ko, H. Jeong, D. Jung and S. Pack. 'Dynamic Split Computing Framework in Distributed Serverless Edge Clouds'. In: *IEEE Internet of Things Journal* (2023).

[165] T. Wang, Y. Liang, X. Shen, X. Zheng, A. Mahmood and Q. Z. Sheng. 'Edge Computing and Sensor-Cloud: Overview, Solutions, and Directions'. In: *ACM Computing Surveys* (2023).

[166] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania *et al.* 'PyTorch distributed: experiences on accelerating data parallel training'. In: *Proceedings of the VLDB Endowment* 13.12 (2020), pp. 3005–3018.

[167] H. Touvron, M. Cord and H. Jégou. 'DeiT III: Revenge of the ViT'. In: *European conference on computer vision*. Springer. 2022, pp. 516–533.

[168] R. Xu, H. Xiang, X. Xia, X. Han, J. Li and J. Ma. 'Opv2v: An open benchmark dataset and fusion pipeline for perception with vehicle-to-vehicle communication'. In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 2583–2589. DOI: 10.1109/ICRA46639.2022.9812038.

[169] Z. Chen, F. Zhong, Q. Luo, X. Zhang and Y. Zheng. 'Edgevit: Efficient visual modeling for edge computing'. In: *International Conference on Wireless Algorithms, Systems, and Applications*. Springer. 2022, pp. 393–405.

[170] K. Wu, J. Zhang, H. Peng, M. Liu, B. Xiao, J. Fu and L. Yuan. 'Tinyvit: Fast pretraining distillation for small vision transformers'. In: *European Conference on Computer Vision*. Springer. 2022, pp. 68–85.

[171] X. Liu, H. Peng, N. Zheng, Y. Yang, H. Hu and Y. Yuan. 'Efficientvit: Memory efficient vision transformer with cascaded group attention'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 14420–14430.

[172] R. Xu, Z. Tu, H. Xiang, W. Shao, B. Zhou and J. Ma. 'CoBEVT: Cooperative Birds Eye View Semantic Segmentation with Sparse Transformers'. In: *Conference on Robot Learning*. PMLR. 2023, pp. 989–1000.

[173] R. Xu, Y. Guo, X. Han, X. Xia, H. Xiang and J. Ma. 'Opencda: an open cooperative driving automation framework integrated with co-simulation'. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE. 2021, pp. 1155–1162. DOI: 10.1109/ITSC48978.2021.9564825.

[174] J.-E. Deschaud. 'KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator'. In: *arXiv preprint arXiv:2109.00892* (2021).

[175] S. Zhu, X. Tian, H. Chen, H. Zhu and R. Qiao. 'Edge collaborative caching solution based on improved nsga ii algorithm in internet of vehicles'. In: *Computer Networks* 244 (2024), p. 110307.

[176] C. Ma, A. Li, Y. Du, H. Dong and Y. Yang. 'Efficient and scalable reinforcement learning for large-scale network control'. In: *Nature Machine Intelligence* 6.9 (2024), pp. 1006–1020.

[177] H. Wu, J. Zhang, Z. Cai, F. Liu, Y. Li and A. Liu. 'Toward energy-aware caching for intelligent connected vehicles'. In: *IEEE Internet of Things Journal* 7.9 (2020), pp. 8157–8166.

[178] E. Commission, D.-G. for Energy, A. Louguet, M. Caspani, D. Pytel, A. Pirlot, M. Faura Rosendo and H. Blanadet. *Assessment of the energy footprint of digital actions and services*. Publications Office of the European Union, 2023. DOI: doi/10.2833/478689.

[179] D. Katare, D. S. Noguero, S. Park, N. Kourtellis, M. Janssen and A. Y. Ding. 'Analyzing and Mitigating Bias for Vulnerable Road Users by Addressing Class Imbalance in Datasets'. In: *IEEE Open Journal of Intelligent Transportation Systems* (2025), pp. 1–1. DOI: 10.1109/OJITS.2025.3564558.

[180] D. Katare, M. Janssen and A. Y. Ding. 'Energy-Aware Adaptive Framework for CAV'. In: *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*. 2025, pp. 626–632. DOI: 10.1109/CCWC62904.2025.10903937.

[181] E. Peltonen, S. Bayhan, D. Bermbach, S. Buschjager, V. Degeler, A. Y. Ding, O. D. Incel, D. Katare, M. B. Kjargaard, S. Leroux *et al.* 'Rethinking Computing Systems in the Era of Climate Crisis: A Call for a Sustainable Computing Continuum'. In: *IEEE Internet Computing* 01 (2025), pp. 1–9.

[182] S. Leroux, D. Katare, A. Y. Ding and P. Simoens. 'Test-time Specialization of Dynamic Neural Networks'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2024, pp. 1048–1056.

[183] D. Katare and M. El-Sharkawy. ‘Real-time 3-d segmentation on an autonomous embedded system: using point cloud and camera’. In: *2019 IEEE National Aerospace and Electronics Conference (NAECON)*. IEEE. 2019, pp. 356–361. DOI: 10.1109/NAECON46414.2019.9057988.

[184] D. Pathak, R. N. Sarangapani, D. Katare, A. S. Gaikwad and M. El-Sharkawy. ‘IOT based solution for level detection using CNN and OpenCV’. In: *Proceedings on the International Conference on Internet Computing (ICOMP)*. 2018, pp. 83–87.

Acknowledgements

A journey is incomplete without companions. I am deeply grateful to those who took this challenging path with me, offering their insight, support, and friendship every step of the way!

I sincerely thank my advisor, Dr. Aaron Ding, for the opportunity to conduct ambitious research within such a supportive and cooperative environment. I am deeply indebted to his wisdom, guidance, and unwavering support throughout my PhD journey, which helped me stay focused on significant problems and consistently strive for high-quality work. I am also grateful for his encouragement to participate in technical events, workshops, and seminars. These valuable engagements allowed me to present our work and connect with senior researchers across academia and industry, providing an external perspective that helped me assess our research and better understand its practical relevance within the community landscape. Collaborating and learning from him has been truly transformative. I extend my heartfelt gratitude to my promoter, Prof. Marijn Janssen, for his steady guidance and insightful feedback throughout my PhD. His strategic vision and deep understanding of the research process helped me stay organized and focused on both short and long-term goals. Our regular discussions brought clarity to each stage of this research and strengthened my approach to proposal development, planning, and evaluation. I am especially thankful for his patience with my tentative timelines, his constructive advice, and his practical perspective, which were essential in shaping the progress of this research. Together, you created an environment of trust, constructive challenge, and encouragement that shaped both my academic and personal growth. I could not have imagined a more ambitious, supportive, and inspiring advisory team. The phrase "Standing on the Shoulders of Giants" feels especially fitting, and I feel truly fortunate to have had that experience.

A special thanks to the current and previous ICT staff members, Ellen Schwencke-Karlas, Fanny Voets, Minaksie Ramsoekh, and Laura Bruns, for their dedicated and efficient support. Their timely assistance with administrative matters, scheduling, and coordination kept many tasks well-organized and ensured a smooth process throughout my PhD. I would like to extend my sincere thanks to my office mates, Aga and Gilang, for the thoughtful discussions and for sharing their knowledge and approaches from their own PhD journeys, which helped me avoid feeling isolated in my own bubble. I also thank my ESS and ICT colleagues: Antra, Sem, Inigo, Anneke, Jolien, Roel, and Mark for sharing their expertise and for being a source of motivation and encouragement

I had the privilege to spend part of my research period at Telefónica R&D in Barcelona, Spain, as part of an industrial internship. The discussions and collaborations with senior researchers and research groups enriched and influenced this work. I am grateful to the research scientists and my collaborators, Nicolas Kourtellis, Eduard Marin, David Solans,

Souneil Park, and Diego Perino, for their guidance and teamwork, which deepened my understanding and provided valuable insights into conducting research within a broader industrial setting. I would also like to thank my current and previous CPI group mates Prachi, Wiebke, Reni, Syafira, Marcus, and Cilia for their feedback and suggestions during our research meetings. Within the group, I am also thankful to visiting researchers Mengying Zhou and Sam Leroux, whose collaborations brought new perspectives and further inspired my research methods and scope.

A special thanks to the APROPOS ITN team, in particular Dr. Alexander Ometov, Prof. Jari Nurmi, and fellow MSCA early-stage researchers Piotr Kluska, Sudam Wasala, Ahmad Mirsalari, Saba Yousefzadeh, Ali Piri, Sepide Saeedi, Lev Denisov, and Jie Lei for making the doctoral schools and ITN meetings engaging and rewarding through their discussions. I also extend my gratitude to my research collaborators from the doctoral training network, Salar Shakibhamedan, Hans Jakob Damsgaard, Antoine Grenier, Zain Taufique, Tiago Troccoli, and Georgios Chatzitsompanis, for their valuable contributions to joint publications and project work packages.

I would like to express my gratitude to Thayaharen, Obinna, Bahador, Yasser, Jianyang, Renatos, and Jet for their support. Your presence played a role in my decision to pursue a PhD in the Netherlands. A special thank you also goes to Niranjan Ravi, Arush Sharma, and Saurabh Kumar, who helped me in research discussions, provided feedback on presentations and research papers, and contributed to my personal and professional growth. I am also thankful to my friends Akash, Durvesh, Harshal, Mitra, Shantanu, Rajas, Irtsam, Onkar, Rishikesh, Siva, Raghavan, Sreeram, Rojin, and Prakriti Jain for adding laughter, balance, and perspective to my life outside of academics. Your conversations and shared moments made this journey lighter and more memorable. Whether near or far, your encouragement always reached me when it mattered most. Lastly, I would like to extend a heartfelt thanks to my family for their constant support of my goals, wishes, and academic aspirations. To my sister, Vijaylaxmi, and brother, Prashant, thank you for keeping me grounded, occasionally roasted, and always reminding me that there's life beyond research (even if you still think 'approximation' and 'deep learning' are fancy research terms). To my mother, Meera, this journey has been a collective dream, and I hope we can fulfill the remaining wishes too. In memory of my late father, Ram Kishore, this PhD is dedicated to you; I wish you could have witnessed it. Your ambitions and thoughts will continue to live through me, as thoughts and ambitions do travel far.

List of Publications

17. D. Katare, D. S. Noguero, S. Park, N. Kourtellis, M. Janssen and A. Y. Ding. ‘Analyzing and Mitigating Bias for Vulnerable Road Users by Addressing Class Imbalance in Datasets’. In: *IEEE Open Journal of Intelligent Transportation Systems* (2025), pp. 1–1. DOI: 10.1109/OJITS.2025.3564558

16. D. Katare, S. Leroux, M. Janssen and A. Y. Ding. ‘Approximating vision transformers for edge: variational inference and mixed-precision for multi-modal data’. In: *Computing* 107.3 (2025), p. 71. DOI: 10.1007/s00607-025-01427-w

15. D. Katare, E. Marin, N. Kourtellis, M. Janssen and A. Y. Ding. ‘ARASEC: Adaptive Resource Allocation and Model Training for Serverless EdgeCloud Computing’. In: *IEEE Internet Computing* 28.6 (2024), pp. 17–27. DOI: 10.1109/MIC.2024.3514670

14. D. Katare, D. Perino, J. Nurmi, M. Warnier, M. Janssen and A. Y. Ding. ‘A survey on approximate edge ai for energy efficient autonomous driving services’. In: *IEEE Communications Surveys & Tutorials* (2023). DOI: 10.1109/COMST.2023.3302474

13. D. Katare, S. Shakibhamedan, N. Amirafshar, N. Taherinejad, A. Jantsch, M. Janssen and A. Y. Ding. ‘Approximation Strategies for Vision Models on Edge Devices: An Accuracy-Efficiency Trade-off’. In: *TechRxiv-Preprint* (2024)

12. D. Katare, M. Janssen and A. Y. Ding. ‘Energy-Aware Adaptive Framework for CAV’. in: *2025 IEEE 15th Annual Computing and Communication Workshop and Conference (CCWC)*. 2025, pp. 626–632. DOI: 10.1109/CCWC62904.2025.10903937

11. D. Katare, M. Zhou, Y. Chen, M. Janssen and A. Y. Ding. ‘Energy-Aware Vision Model Partitioning for Edge AI’. in: *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*. SAC ’25. 2025, pp. 1–8. DOI: 10.1145/3672608.3707792

10. D. Katare and A. Y. Ding. ‘Energy-efficient edge approximation for connected vehicular services’. In: *2023 57th Annual Conference on Information Sciences and Systems (CISS)*. IEEE. 2023, pp. 1–6. DOI: 10.1109/CISS56502.2023.10089724

9. D. Katare, N. Kourtellis, S. Park, D. Perino, M. Janssen and A. Y. Ding. ‘Bias detection and generalization in AI algorithms on edge for autonomous driving’. In: *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. IEEE. 2022, pp. 342–348. DOI: 10.1109/SEC54971.2022.00050

8. H. J. Damsgaard, A. Grenier, D. Katare, Z. Taufique, S. Shakibhamedan, T. Troccoli, G. Chatzitsompanis, A. Kanduri, A. Ometov, A. Y. Ding *et al.* ‘Adaptive approximate computing in edge AI and IoT applications: A review’. In: *Journal of Systems Architecture* (2024), p. 103114. DOI: https://doi.org/10.1016/j.sysarc.2024.103114

7. E. Peltonen, S. Bayhan, D. Bermbach, S. Buschjager, V. Degeler, A. Y. Ding, O. D. Incel, D. Katare, M. B. Kjargaard, S. Leroux *et al.* 'Rethinking Computing Systems in the Era of Climate Crisis: A Call for a Sustainable Computing Continuum'. In: *IEEE Internet Computing* 01 (2025), pp. 1–9

6. S. Leroux, D. Katare, A. Y. Ding and P. Simoens. 'Test-time Specialization of Dynamic Neural Networks'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2024, pp. 1048–1056

5. D. Katare and M. El-Sharkawy. 'Embedded system enabled vehicle collision detection: an ANN classifier'. In: *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE. 2019, pp. 0284–0289. DOI: 10.1109/CCWC.2019.8666562

4. D. Katare and M. El-Sharkawy. 'Real-time 3-d segmentation on an autonomous embedded system: using point cloud and camera'. In: *2019 IEEE National Aerospace and Electronics Conference (NAECON)*. IEEE. 2019, pp. 356–361. DOI: 10.1109/NAECON46414.2019.9057988

3. D. Katare and M. El-Sharkawy. 'Autonomous Embedded System Enabled 3-D Object Detector:(with Point Cloud and Camera)'. In: *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. IEEE. 2019, pp. 1–6. DOI: 10.1109/ICVES.2019.8906442

2. D. Katare and M. El-Sharkawy. 'Collision warning system: embedded enabled (RTMaps with NXP BLBX2)'. In: *2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*. IEEE. 2018, pp. 1–6. DOI: 10.1109/ISSPIT.2018.8705101

1. D. Pathak, R. N. Sarangapani, D. Katare, A. S. Gaikwad and M. El-Sharkawy. 'IOT based solution for level detection using CNN and OpenCV'. in: *Proceedings on the International Conference on Internet Computing (ICOMP)*. 2018, pp. 83–87

About the author

Dewant Katare was born in Chhattisgarh, India, in 1992. He received his Bachelor of Science in Electronic and Electrical Engineering in 2016 under the supervision of Dr. Aleksandar Andreski and Dr. Ruud Steenwelle, and a Master of Science in Electrical and Computer Engineering from Purdue University in 2019, supervised by Prof. Mohamed El-Sharkawy. In 2021, he began his doctoral research as an early-stage researcher at the Faculty of Technology, Policy and Management, Delft University of Technology (TU Delft), within the Information and Communication Technology (ICT) section, as part of the Marie Skodowska-Curie Innovative Training Network project APROPOS: Approximate Computing for Power and Energy Optimization.

His PhD research focused on exploring energy-aware models and computing strategies for computer vision systems deployed on distributed edge devices, with applications in automated driving services. His work explored model approximation techniques, distributed and collaborative deployment strategies, and energy-efficient model partitioning frameworks for sustainable Edge AI applications, while also investigating the correlation between model behavior, system performance, and hardware-specific metrics.

During his doctoral studies, he was a visiting researcher at Telefónica Research in Barcelona, Spain, within the research group of Dr. Diego Perino and Dr. Nicolas Kourtellis, where he worked on understanding AI model learning patterns, bias detection and mitigation strategies for machine learning models used in autonomous driving, as well as cost-efficient model resource allocation and partition mechanisms for distributed deployment. During his doctoral research, he has published in peer-reviewed journals and conferences, including IEEE Communications Surveys & Tutorials, IEEE Open Journal of Intelligent Transportation Systems, Springer Nature Computing, Elsevier Journal of Systems Architecture, IEEE Internet Computing, IEEE Conference on Information Sciences and Systems, and the ACM Symposium on Applied Computing among others.

Before starting his PhD, Dewant gained professional and research experience at NXP Semiconductors (Netherlands), Endress+Hauser (USA), and the Anthropocenes Network (USA), where he worked on microcontrollers, automotive vision processors, embedded systems, sensor networks, and real-time vision applications using deep learning. His research interests include distributed machine learning, computer vision, approximate computing, and sustainable AI systems.

Propositions

accompanying the dissertation

ADAPTIVE ENERGY-AWARE FRAMEWORK FOR CONNECTED VEHICLE SERVICES

by

Dewant KATARE

1. Software-level approximation techniques are able to balance the computational demands of AI models in connected vehicles applications without sacrificing models accuracy. (this thesis)
2. Metric-specific learning for AI models provides model learning patterns that improve algorithm bias mitigations, model interpretability, and fairness. (this thesis)
3. Collaborative edge computing mechanisms improve resource allocation in heterogeneous memory and compute settings while reducing the energy consumption of applications. (this thesis)
4. AI can only support complex socio-technical systems when two operations of human understanding are integrated, i.e. intuition and deduction.
5. The deployment of AI models on heterogeneous edge systems requires adaptive algorithms that consider computing constraints and device-specific capabilities.
6. For real-world applications using AI models, robustness, efficiency, and reliability must be considered as fundamental evaluation criteria alongside accuracy.
7. Integrating energy consumption and carbon footprint assessments into the AI development life cycle is essential for creating sustainable practices in AI research and development.
8. A researcher's ability to explain the fundamental and mathematical components of their developed AI model is inversely proportional to the model's complexity.
9. There is no statistical or probabilistic correlation between coffee consumption and the speed of solving research problems.
10. The probability of a computer or device crash increases exponentially as the deadline for paper submission and application demonstration approaches.

These propositions are regarded as opposable and defendable, and have been approved as such by the promotors, prof. dr. ir. M.F.W.H.A. Janssen and dr. Y. Ding.

This thesis presents an adaptive framework with submodules that enable model partitioning, resource allocation, and approximation strategies for AI models deployed across the vehicle–edge–cloud computing hierarchy. The framework addresses the high computational and energy demands of modern data-intensive vehicular services by distributing workloads intelligently and optimizing precision levels in real time.

As connected and automated driving systems increasingly rely on deep learning, efficient deployment on embedded and battery-powered devices remains a significant challenge. This research overcomes this by proposing software-level mechanisms that enhance energy efficiency while maintaining reliable task performance. The framework dynamically adapts computation precision and allocates resources across heterogeneous hardware (CPUs, GPUs, and AI accelerators) by using the inherent error tolerance of AI models.

Evaluations of deep neural networks and transformer-based vision models shows a balanced trade-off, achieving up to 40% energy savings with a loss of less than 7% in accuracy. This work contributes a scalable and energy-aware software solution for sustainable AI deployment in connected vehicle ecosystems, advancing energy-efficient computing for next-generation intelligent transportation systems.