Energy-Efficient Acceleration for Autonomous Robotics

by

Deval Shah

B.Engineering, Maharaja Sayajirao University of Vadodara, 2014

M.Technology, Indian Institute of Technology, Bombay, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES

(Electrical and Computer Engineering)

The University of British Columbia

(Vancouver)

May 2024

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Energy-Efficient Acceleration for Autonomous Robotics

submitted by **Deval Shah** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy** in **Electrical and Computer Engineering**.

Examining Committee:

Tor M. Aamodt, Professor, Department of Electrical and Computer Engineering, THE UNIVERSITY OF BRITISH COLUMBIA
Supervisor

Mieszko Lis, Professor, Department of Electrical and Computer Engineering, THE UNIVERSITY OF BRITISH COLUMBIA
Supervisory Committee Member

Prashant J. Nair, Assistant Professor, Department of Electrical and Computer Engineering, THE UNIVERSITY OF BRITISH COLUMBIA
Supervisory Committee Member

Septimiu E. Salcudean, Professor, Department of Electrical and Computer Engineering, THE UNIVERSITY OF BRITISH COLUMBIA
University Examiner

Ian Mitchell, Professor, Department of Computer Science, THE UNIVERSITY OF BRITISH COLUMBIA
University Examiner

Abstract

Autonomous robots have great potential to improve our day-to-day lives. Hence, their demand for real-time and safety-critical tasks is increasing, including medical care, home assistance, and autonomous driving. An autonomous robot system consists of several latency-sensitive computation tasks. Further, a crucial aspect of computing for autonomous robots is energy efficiency, as computation can contribute to a significant fraction of the total power consumption. With the recent advancements in machine learning, algorithms for autonomous robots' operation are explored extensively at the software level. However, their architectural implications are not explored in detail.

This thesis explores algorithm and hardware design for energy-efficient computation acceleration of perception and planning tasks in autonomous robotics. It first proposes a motion planning hardware accelerator, MPAccel. MPAccel consists of a Spatially-Aware Scheduler and Collision Prediction Units. The proposed scheduler and predictor exploit the physical-spatial locality of a robot's positions in the environment to reduce redundant computation. Further, MPAccel proposes a hierarchical collision detection approach and corresponding hardware accelerator. MPAccel enables real-time sampling-based motion planning for a 7-DOF robotic arm with less than 1ms latency and 3.5W power consumption. The second part of the thesis focuses on regression networks, as they are used in autonomous robot perception and planning. It explores the design space of regression by binary classification and proposes Binary-Encoded Labels (BEL). This thesis further analyzes the properties of suitable label encodings and proposes manually designed label encodings. It also proposes an end-to-end training approach to learn label encodings and network parameters together for a given task. BEL reduces regression error by

10.9% compared to direct regression, enabling the use of smaller and/or sparser networks for compute- and energy-efficient execution. Further, BEL-based regression networks are explored for smaller and sparser neural motion planners, reducing neural planning computation by a factor of 11.4×. The third part of the thesis explores the impact of soft errors on collision detection hardware accelerators in motion planning. It proposes an application-specific reliability metric, Collision Exposure Factor, and demonstrates its use for faster fault characterization and low-overhead selective error mitigation.

Lay Summary

This dissertation aims to provide energy-efficient acceleration of various computation tasks in autonomous robots with algorithm and hardware design. It focuses on robot motion planning and deep regression neural networks. This dissertation explores how the physical spatial locality of different positions of a robot in its environment can be leveraged to reduce computation and proposes a motion planning hardware accelerator. Further, it explores an application-specific reliability metric for low-overhead error mitigation in robotic hardware accelerators. Finally, it explores the use of binary classification to solve regression problems and provides significant improvement in regression accuracy for dense and sparse regression networks.

Preface

This section lists all published manuscript completed during the program. Further, it describes how these works are incorporated in the dissertation, providing more details of my contributions.

The following is a list of my publications incorporated in this dissertation in chronological order.

[C1] Deval Shah, Zi Yu Xue, and Tor M. Aamodt. Label Encoding for Regression Networks [223]. In *International Conference on Learning Representations*, April 2022. URL https://openreview.net/pdf?id=8WawVDdKqlL.

[C2] Deval Shah and Tor M. Aamodt. Learning Label Encodings for Deep Regression [221]. In *International Conference on Learning Representations*, May 2023. URL https://openreview.net/pdf?id=k60XE_b0Ix6.

[C3] Deval Shah, Ningfeng Yang, and Tor M. Aamodt, Energy-Efficient Realtime Motion Planning [225], in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589092

[C4] Deval Shah, Zi Yu Xue, Karthik Pattabiraman, and Tor M. Aamodt, Characterizing and Improving Resilience of Accelerators to Memory Errors in Autonomous Robots [224], *ACM Transactions on Cyber-Physical Systems*, Oct 2023. ISSN 2378-962X. doi:10.1145/3627828. URL https:doi.org/10.

1145/3627828.

[C5] Deval Shah and Tor M. Aamodt, Collision Prediction for Robotics Accelerators [222], in *Proceedings of the 51st Annual International Symposium on Computer Architecture*, ser. ISCA '24. Just Accepted.

Chapter 1 The motivation from [C1]-[C5] is paraphrased in this chapter.

Chapter 2 The portion of this chapter describing motion planning and collision detection is taken from [C3], [C4], and [C5]. Portions of this chapter related to deep regression networks are taken from [C1] and [C2]. The description of soft-error resilience is taken from [C4].

Chapter 3 A portion of this chapter related to the Spatially Aware Scheduler, Cascaded Early-Exit Collision Detection Unit, and MPAccel has been published as [C3]. I was the lead researcher for this work and conducted research, design and implementation of microarchitectural simulators, data interpretation, and manuscript writing under the supervision of Dr. Tor M. Aamodt. Mabel Wang helped with the initial experimental setup for the MPNet implementation code provided by the authors. Ningfeng Yang worked on the RTL implementation of all hardware components and the CPU and GPU evaluation of the collision detection algorithm under Dr. Tor M. Aamodt's and my supervision. A section of this chapter on Collision Prediction has been accepted for publication in a separate paper [C5]. I performed the research, design, implementation, data interpretation, and manuscript writing under the supervision of Dr. Tor M. Aamodt.

Chapter 4 A portion of this chapter has been published as [C1]. I was the lead researcher for this work. I conducted research, design, implementation, result interpretation, and manuscript writing under the guidance of Dr. Tor M. Aamodt, with inputs from Zi Yu Xue. Zi Yu Xue implemented benchmarks related to age estimation and end-to-end learning of self-driving car steering for evaluation and also helped write corresponding sections in the manuscript. A portion of this chapter related to label encoding learning has been published as [C2]. A manuscript is under preparation that includes the last portion of this chapter

related to label encoding for neural path planning. I was the lead researcher for both of these works and carried out research, design and implementation, result interpretation, and manuscript writing under the guidance of Dr. Tor M. Aamodt.

Chapter 5 A version of this chapter has been published as [C4]. I was the lead researcher responsible for conducting research, design and implementation, data interpretation, and manuscript writing under the guidance of Dr. Tor M. Aamodt and Dr. Karthik Pattabiraman, with input from Zi Yu Xue. Zi Yu Xue worked on RTL synthesis and RTL-fault injection. He also performed a literature survey of existing fault characterization tools, ECC techniques, and functional safety and helped with writing corresponding sections in the manuscript.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AABB	Axis Aligned Bounding Box
AE	Age Estimation
ASIC	Application-Specific Integrated Chips
BEL	Binary-Encoded Labels
BRP	Binary Recursive Policy
CDC	Collision Detection Circuits
CDF	Collision Detection Failure rate per billion hours
CDM	Collision Detection Module
CDU	Collision Detection Unit
CECDU	Cascaded Early-exit Collision Detection Unit
CEF	Collision Exposure Factor
COPU	Collision Prediction Unit
CPU	Central Processing Unit
CSP	Coarse-Step Policy
DOF	Degree of Freedom
FI	Fault Injection
FIT	Failure-in-time, number of failures per billion hours
FLD	Facial Landmark Detection
FPGA	Field Programmable Gate Array

GPU	Graphics Processing Unit
LEL	Label Encoding Learning
LFH	Landmark-free Facial Headpose Estimation
MCSP	Multi-motion Coarse-Step Policy
MPA	Motion Planning Accelerator, a generic term used for a motion planning accelerator
MPACCEL	Motion Planning Accelerator
NP	Naive Parallel
OBB	Oriented Bounding Box
OOCD	OBB-Octree Collision Detector
PN	PilotNet
PRM	Probabilistic Roadmaps
RLEL	Regularized Label Encoding Learning
SAS	Spatially Aware Scheduler
SDC	Silent Data Corruption
SDC-C	Silent Data Corruption leading to false-negative collision detection output
SIL	Safety Integrity Level

Acknowledgments

First and foremost, I am extremely grateful to my PhD advisor, Dr. Tor M. Aamodt, for his continuous support and invaluable guidance. His mentorship has been instrumental in shaping my research journey. I have learned and am still learning several aspects of research under him, including research direction and ideas formation, research paper writing, research presentation, and mentoring. He always prioritized my goals and learning and guided me at each step of this journey. He has been a terrific mentor to me, and the knowledge and skills I have gained under his mentorship will be invaluable throughout my career.

I would also like to thank my supervisory committee members, Dr. Mieszko Lis and Dr. Prashant Nair, for their valuable feedback on my work. I would also like to thank my collaborators, Dr. Karthik Pattabiraman, Zi Yu Xue, and Ningfeng Yang, for their help and feedback on work included in this dissertation.

A special thanks to my colleagues and fellow research graduates at the Computer Architecture lab, Dr. Tayler Hetherington, Dr. R. David Evans, Dr. Ayub Gubran, Maria Lubeznov, Aamir Raihan, Amruth Sandhupatla, Negar Goli, Francois Demoullin, Tommy Chou, Jonathan Lew, LuFei Liu, Mohammadreza Saed, who have helped in various aspects throughout this journey. Their feedback has been invaluable in improving my research work.

I would like to acknowledge the funding sources that made my research possible: Four-Year Fellowship, NSERC Discovery Grant, and the NSERC Strategic Research Network for Computing Hardware for Emerging Intelligent Sensing Applications (NSERC COHESA).

On a personal note, I am incredibly grateful to my brother, Rahil Shah, for inspiring me to pursue this journey. My parents, Ashok Shah and Nayana Shah, deserve

my heartfelt thanks for their unwavering support and encouragement throughout my life. They always prioritized my needs and education and shaped my childhood to become who I am. I am also grateful to my sister-in-law, Poyani Sheth, for her support.

Last but not least, my deepest gratitude goes to my husband, Pranav Sampara. We embarked on this journey together, and his presence, support, and understanding have been invaluable. Thank you for always being there for me.

Chapter 1

Introduction

Autonomous robots have huge potential for improving our day-to-day life in various aspects. The field of autonomous robotics is at a very exciting point, and there is increasing interest in deploying autonomous robots for different tasks. Robotics for medical care, elderly care, task completion in hazardous environments, and home assistance can directly impact our society. Consequently, the field of autonomous robots is advancing rapidly. For example, the size of the autonomous robots market is expected to grow by 4$\times$ from 2022 to 2030 [26, 116].

Recent developments in the field of artificial intelligence and computation have made the deployment of autonomous robots for a wide range of tasks a near-future possibility. Artificial intelligence can push the capability of robots by providing powerful means for perception and planning. For example, computer vision, which includes several tasks of perceiving different information about the environment from a visual input such as images or videos, has seen significant development in the last decade with the progress made in deep learning. Recent leaps in deep learning, including convolution networks, transformers, and, more recently, diffusion networks, have resulted in neural networks capable of achieving high accuracy in several computer vision tasks. These developments have made it possible to understand the characteristics of a scene and enable rich perception modules useful for an autonomous robot [31, 72, 243, 264, 276].

An autonomous robot is a complex system with several sensors, actuators, and compute modules. Earlier, robot deployment was carried out in a limited and

constrained scope, where a robot was used for very specific tasks. However, as robotics and artificial intelligence fields advance, approaches capable of enabling these autonomous robots to perform a wide range of complex tasks with minimal human intervention are explored extensively at the algorithm level. However, with these advances, the future generation of autonomous robots is expected to perform complex tasks in a wide range of environments, requiring higher computation power for a robot's perception, reasoning, planning, and control. Thus, studying the architectural implications of different tasks involved in an autonomous robot's operation and development of a suitable computing platform is essential, and is a bottleneck in its deployment in several cases [171, 247].

1.1 Computation Tasks in Autonomous Robots

An autonomous robot can sense and perceive the environment, make decisions, and perform actions to achieve its goals without explicit human intervention. An autonomous robot is a complex system with several modules, including sensors, compute units, and actuators. It uses one or more sensors to receive information about the environment. Controlling different mechanical parts of an autonomous robot (e.g., limbs of a robot) is carried out by sending control signals to its actuators.

Several computation tasks are involved between sensing environmental information and sending control signals to actuators [79]. Figure 1.1 represents different stages and some representative examples of computation tasks needed for the operation of an autonomous robot to achieve a specific goal. In the first stage, one or more sensors, such as a camera, LiDAR, microphone, GPS, and Radar, are used to get observations of the environment. The set of sensors used in an autonomous robot is a design choice, and it depends upon several factors, such as tasks the robot needs to perform, expected environmental scenarios, cost budget, and the robot's topology. Next, the robot perceives the environment from the sensors' output data to decide the following action. It converts the sensors' output data to higher-level semantics to do so. For example, an autonomous robot must perform localization and mapping to navigate to a goal position. It performs occupancy mapping to find which parts of the environment are occupied by obstacles for obstacle avoidance while moving. It can perform object detection on the camera's output and convert the input image to

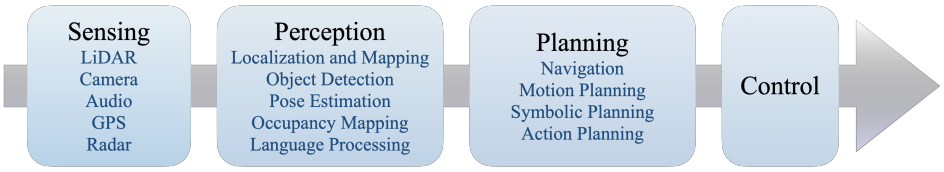


Figure 1.1: Computation tasks in autonomous robots.

a list of different objects and their location in the image. Perception tasks performed by a robot depend upon what information it needs to perform planning. For example, several path-planning algorithms for autonomous vehicles require lane information from the perception module. In the planning stage, the perceived information is used to decide the next step of the robot to achieve the overall goal in the last stage, which consists of different planning tasks. Action planning mainly decides what actions the robot should take to finish a task. Motion planning decides how a robot should move through space to reach the end position. A smarter robot can perform more sophisticated tasks, such as reasoning before action planning, to achieve higher autonomy. Finally, the control module computes the signals sent to actuators to follow the path/trajectory determined by the planning module.

1.2 Computation Challenges in Autonomous Robots

The computation pipeline in an autonomous robot consists of many tasks, and such a system requires a heterogeneous computing system composed of several general-purpose and/or application-specific compute cores. Integration of many dependent computation tasks raises several architectural challenges, including scheduling, interface, communication, and acceleration of individual computation tasks. Given the huge scope of this problem, in this dissertation, I focus on two major challenges in autonomous robotics: latency and energy consumption of computation kernels related to planning and perception.

True autonomy requires the capability of safely navigating and responding in real time to a dynamically changing environment. This requirement imposes strict requirements of the end-to-end latency of computation tasks (in the range of ms). Several computation tasks are performed between sensing and actuating in autonomous robots. Thus, latencies of different computation tasks accumulate and

affect the response time of a robot to a dynamic environment, which constrains the kind of environment a robot can operate in. For example, sensing-to-brake latency for autonomous vehicles or drones determines the maximum safe speed for the vehicle [135]. Further, the problem is exacerbated for robots with a high degree of freedom (DOF), as the complexity of planning algorithms increases exponentially with DOFs. These computation tasks' strict real-time latency requirements are beyond CPUs' capability. Thus, several compute acceleration approaches for different algorithms using GPUs, FPGAs, and ASICs have been studied by prior works [8, 11, 18, 76, 122, 153, 171, 173, 226, 233, 278].

Another challenge of real-time acceleration for autonomous robotics is the energy consumption of the computing platform. Often, acceleration of planning and perception for autonomous robots for real-time latency comes at the expense of high energy consumption. For example, the mechanical part of the Jaco2 robot consumes 35W power. Prior works on GPU or ASIC-based acceleration of motion planning results in comparable or higher power consumption compared to the mechanical parts [18, 173]. The problem becomes more significant for mobile robots with limited batteries, as the power consumption of the computation system can significantly reduce a mobile robot's operation time with a single charge. Thus, developing acceleration approaches for computation tasks in autonomous robots while lowering energy consumption is crucial.

Overall, the computation demand in an autonomous robot increases with the complexity of its tasks and the amount of intelligence the robot needs to possess. The latency and energy efficiency provided by the compute acceleration system envelopes the "capability or intelligence" achievable by an autonomous robot system. By building faster and more energy-efficient acceleration approaches, we can push the boundary of the capabilities of autonomous robots, and that is the key motivation behind the works presented in this dissertation.

1.3 Thesis Statement

This dissertation explores the computation tasks in autonomous robot's perception and planning and makes contributions towards energy-efficient acceleration of these computation tasks using hardware and/or algorithm optimizations.

The first part of this dissertation focuses on hardware acceleration of a key task performed by autonomous robots: motion planning. A dominant part of motion planning is collision detection between possible robot poses and environmental obstacles. The key insight used for this contribution is that the physical spatial locality of objects and a robot's poses can be exploited to develop application-specific algorithm-hardware co-optimization approaches that not only accelerate motion planning and collision detection but also improve energy efficiency by reducing redundant computation.

The second part of my research focuses on deep regression networks. The motivation behind focusing on regression networks is that these networks are prominently used in robotics. For example, several computer vision tasks are relevant to robot perception, including pose estimation, lane detection, and occupancy mapping. Further, recent neural planners for motion planning and end-to-end approaches for robotics operations predominantly use regression networks. While several approaches have focused on classification networks and acceleration of these networks using algorithm-hardware co-optimization (e.g., model compression and sparse accelerators), few works focus on a generalized approach for regression networks. With this motivation, we focus on a generic approach for regression by binary classification to improve the accuracy of regression tasks in robotics. This improvement in regression networks can be exploited to devise energy-efficient and faster algorithms for autonomous robots. We show this by combining model compression approaches with the proposed regression approach and demonstrate its impact on inference computation in computer vision and motion planning tasks for autonomous robots.

Finally, the third part focuses on the safety implications of hardware acceleration in autonomous robots. Application-specific hardware accelerators are designed with memory and compute systems catered to the application and often dedicate a large fraction of the on-chip area to storage structures, making it more prone to soft-error-induced faults. There is a lack of domain-specific fault characterization approaches for autonomous robot hardware accelerators. In this work, we focus on collision detection hardware accelerators for autonomous robots and analyze the impact of soft errors on safety violations, an important aspect of certifiable safety. We find that several accelerators dedicate a large area to spatial representation-related data structures. Based on this insight, we propose an application-specific reliability

metric, Collision Exposure Factor, and show the correlation between this metric and safety violation probability for four collision detection accelerators. We further use the proposed reliability metric to devise faster fault characterization and cost-effective error mitigation techniques.

1.4 Contributions

The following is a list of contributions made in this dissertation.

- It analyzes the impact of parallelization on motion planning acceleration and demonstrates the use of the physical spatial locality of the robot's positions to reduce the computation performed during motion planning. It proposes a Spatially-Aware Scheduler, Collision Prediction Unit, and Cascaded Early-Exit Collision Detection Unit for motion planning. It introduces MPAccel, a motion planning accelerator consisting of the proposed scheduler and collision detection units. Our evaluation suggests that MPAccel can achieve real-time motion planning for a 7-DOF robotic arm with $3.5W$ power consumption.

- It introduces a generalized framework for regression by binary classification and proposes a taxonomy of different design aspects in this framework. It systematically studies this design space and proposes properties of different design choices suitable for regression. It proposes several hand-crafted label encodings (BEL) and decoding functions for regression by binary classification and shows 9.9% improvement in regression error over direct regression. It further establishes the importance of systematically searching this design space for a given problem setup.

- It proposes RLEL, a label encoding learning approach using regularizers to search suitable label encoding and decoding functions for a given regression benchmark. RLEL results in 10.9% improvement over direct regression. It evaluates the impact of model compression using unstructured sparsity on direct regression, BEL, and RLEL. It shows that label encodings for regression result in lower degradation in accuracy with sparsity, making it more suitable for energy-efficient acceleration.

- It proposes a neural path planner using label encodings (BEL). The proposed BEL neural planner results in a higher or comparable motion planning success rate compared to a regression-based neural planner with $11.8\times\%$ reduction in computation using a smaller and sparser model.

- It analyzes the impact of soft errors on collision detection hardware accelerators for Probabilistic Roadmaps-based motion planning. It proposes Collision Exposure Factor (CEF), an application-specific reliability metric. It proposes a CEF-aware fault characterization approach that reduces FI time $24,000\times$ compared to complete FI and finds vulnerable bits in the accelerator memory for selective error mitigation.

1.5 Organization

The rest of the dissertation is organized as follows:

- Chapter 2 provides relevant background information for motion planning, regression in deep learning, and soft-error fault characterization and mitigation.

- Chapter 3 gives details of our work on motion planning acceleration. First, it provides a brief background and terminology of collision detection in motion planning. This chapter details three main contributions made towards motion planning acceleration and provides the microarchitectural implementation of these blocks. Further, this chapter introduces MPAccel, a motion planning accelerator consisting of proposed modules. It provides a detailed evaluation of the proposed approaches.

- Chapter 4 introduces the proposed framework for using label encodings for regression networks. It provides our analytical study, proposed properties of different design aspects of this framework, and manually designed functions. It further gives details of the proposed end-to-end approach to learning label encodings and network parameters for a given benchmark. It provides a detailed evaluation of the proposed approaches. Finally, it explores the impact of sparsity on label encoding-based regression networks for different computer vision and path planning tasks for energy-efficient acceleration.

- Chapter 5 analyzes the impact of soft errors on collision detection hardware accelerators. It provides a brief background of accelerators studied in this work. It introduces an application-specific reliability metric, Collision Exposure Factor (CEF), and proposes CEF-aware fault characterization and error mitigation approaches. It provides a detailed evaluation of the proposed approaches.

Chapter 2

Background

This chapter provides relevant background for this dissertation. First, it provides a background of robotic motion planning and presents relevant details on sampling-based and learning-based motion planning approaches used in this work (Section 2.1). Further, it provides background on collision detection for robotics, one of the most compute-intensive kernels performed during motion planning (Section 2.1.3). Section 2.2 provides background on regression networks, label encodings in deep learning, and encoding design approaches. Finally, this chapter provides relevant background on soft-error fault characterization, error mitigation, and functional safety for autonomous robots in Section 2.3.

2.1 Robot Motion Planning

Motion planning is a fundamental problem in autonomous robots. The objective of motion planning is to compute a collision-free path for a robot between a given pair of start and goal positions. Motion planning is key to the many tasks performed by autonomous robots, including navigation, object manipulation, footstep planning, and full-body movement. It has been an area of study since the 1970s [128, 141], and is today one of the key research topics in robotics. Figure 2.1 represents an example of motion planning for a robotic arm Jaco2 [132] in the presence of environmental obstacles. Below, we provide details on motion planning and collision detection relevant to contributions made in this dissertation. More detailed information about

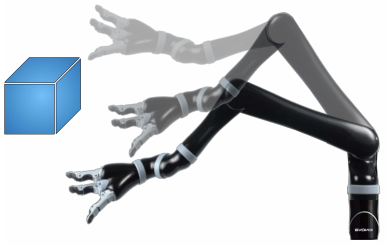
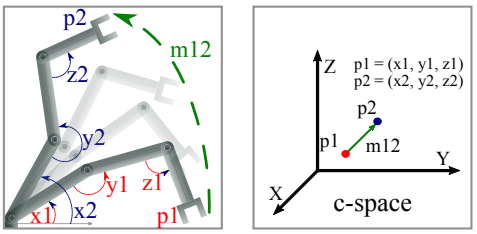


Figure 2.1: Motion planning example for Kinova Jaco2 robotic arm.



(a) **(b)**

Figure 2.2: (a) Spatial poses and motion of a 2D robot with three DOFs (x, y, z) in the physical space, and (b) represents these poses and motion in the robot's C-space.

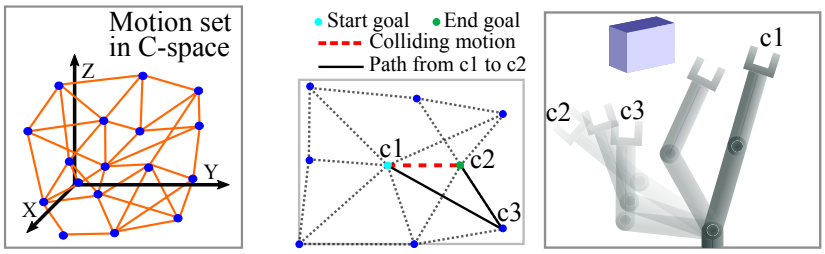
the problem of motion planning and relevant sampling-based motion planning algorithms can be found in [140] (Chapter 3-5) and [217].

Configuration Space (C-space): It is common to phrase a robotic motion planning problem in the configuration space (C-space) of the robot. The C-space of a robot has the same dimensions as its degrees of freedom (DOFs), where each dimension represents the range of values for a DOF (e.g., the angle of a rotational joint). Figure 2.2a represents a 3-DOF robot, and Figure 2.2b represents its C-space. A point with coordinates (x, y, z) in the C-space represents a pose/position of the robot, represented by angles of three joints (x, y, z). Thus, the spatial pose of a robot in the physical space is uniquely identified with a point in the C-space. Points $p1$ and $p2$ in Figure 2.2b represent two different poses taken by the robot in the physical space (Figure 2.2a). The straight line between $p1$ and $p2$ in the C-space (e.g., $m12$) corresponds to a short motion between corresponding poses in the physical space.

2.1.1 Sampling-based Motion Planning

A motion planning problem is typically solved in the C-space of the robot. The C-space of a robot can be divided into two disjoint sets C_{free} and C_{obs}. Here, C_{free} represents the set of all points in the C-space, such that corresponding poses are collision-free for a given environmental scenario (i.e., the position of obstacles). Similarly, C_{obs} represents the set of all points in the C-space, such that the corresponding robot poses are in collision with environmental obstacles. Motion planning between a start and goal configuration can be framed as a problem of finding a path between two configurations such that each pose assumed during this path is in C_{free}. This ensures that all poses taken by the robot during a motion are collision-free. Finding the C_{free} set for a robot and environmental scenario is the crux of the motion planning problem. However, finding this C_{free} set for a given setup is computationally intensive. Further, the complexity of finding this set increases exponentially with the dimension of the C-space, i.e., a robot's DOFs. Hence, sampling-based motion planning algorithms are widely used for motion planning.

It is computationally hard to find the entire set C_{free} for a given robot and environmental scenario. However, the task of finding if a point (pose in physical space) or a line/edge in the C-space (short motion in physical space) are in C_{free} or not is relatively less compute-intensive. For sampling-based motion planning, the C-space is sampled coarsely to find nodes and edges between nearby nodes such that these nodes (robot poses) and edges (i.e., robot's motion between two poses) are in C_{free}. Once such a sparse graph in the C-space is constructed, motion planning between two nodes can be performed using a path search algorithm such as Dijkstra. Figure 2.3a represents such a graph for a robot with 3 DOFs. Figure 2.3b represents the path found between the c1 and c2 in the C-space (left) and physical space (right) using a sampling-based motion planning approach. Several sampling-based motion planning approaches exist that use different strategies for sampling the C-space for building this graph [73, 74, 127, 139, 282]. For example, in the probabilistic roadmap (PRM) [127], a widely used sampling-based motion planning approach, the C-space is sampled randomly. Each point is connected to nearby neighbors with edges. These randomly sampled points and edges in C-space are checked for collision. Collision-free points and edges are used to construct a



(a) **(b)**

Figure 2.3: (a) shows a motion set for a robot with 3 DOF in its C-space. (b) Motion set for a robot with two DOFs in the C-space (left). The figure highlights a path between c1 and c2 in the presence of an obstacle in C-space (left) and physical space (right).

graph G such that all nodes and edges in the graph are in C_{free}. More recently, informed sampling approaches have been proposed. GNNMP uses a graph neural network to take the information about the environment and start/goal positions to give sampled points and edges that are more likely to connect start and end positions and also are in C_{free}. This reduces the computation required for building the graph while increasing the probability of finding a good path between the start and goal points. Different algorithms also interleave the process of graph building and path search [21, 139, 198] until a collision-free path between given start and goal configurations is found. For example, in LazyPRM [21], a small graph is built without a collision check, and a path search between the start and goal node is performed. Collision check is performed if a path is found between the start and the goal. In the case of collision, the graph is modified, and the path search is repeated. If a path between two nodes is not found, then the graph is further expanded for path search. This process continues until a collision-free path is found.

In sampling-based motion planning approaches, the probability of finding a near-optimal path between two poses/configurations, if it exists, increases with the number of nodes and edges in the constructed graph in the C-space. Sampling-based motion planning approaches are often evaluated using motion planning success rate and the quality of motion planning. Here, the success rate specifies the capability of finding a path if one exists. Motion planning quality specifies how close the path is to the optimal path between two poses. Due to the computational advantages of

these motion planning approaches, sampling-based motion planning approaches are widely adapted for high-DOF robotic systems.

Probabilistic Roadmaps

Probabilistic Roadmaps [127] is a widely used sampling-based motion planning approach. Probabilistic motion planning consists of two phases. In the preprocessing stage, a graph consisting of collision-free poses and motions called a *motion set* is constructed in the robot's C-space. Figure 2.3a and Figure 2.3b give examples of motion sets for robots with three and two DOFs, respectively. The nodes in the graph correspond to the robot's spatial poses, and an edge between close-by nodes represents motion generated by a local planner (e.g., linear interpolation between two poses) from one pose to another. In the query phase, a path search algorithm (e.g., Dijkstra's algorithm) is used to find a path between given start and end poses using the precomputed motion set. A path consists of one or more motions from the motion set, as shown in Figure 2.2b. The probabilistic roadmap is a multi-query method, i.e., the same motion set is used to perform motion planning for multiple start and goal poses.

Leven and Hutchinson [144] proposed a motion planning approach based on probabilistic roadmaps for a dynamic environment and is also used by Motion Planning Accelerators (MPAs) [95, 153, 172, 173, 226, 278]. In this approach, the motion set is generated for an obstacle-free environment. At runtime, collision detection is performed to find collision-free poses and motions in this set for a given placement of obstacles. The path search module considers collision-free motions to find a feasible path between the start and end pose in the resulting *collision-free* motion set graph. For example, in Figure 2.3b, an obstacle in the robot's environment makes some of the motions in the motion set flagged as "colliding motion", which are avoided by the path search module to find a path between c1 and c2.

2.1.2 Learning-based Motion Planning

Learning-based motion planning is an active area of research in robotics. Learning-based motion planning approaches use deep neural networks to replace one or

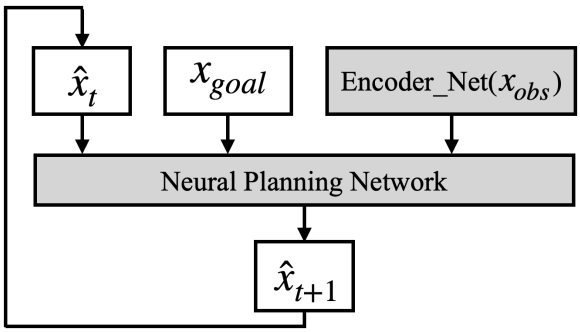


Figure 2.4: Neural Planning Network for motion planning [198].

more modules in traditional motion planning algorithms or use neural networks for end-to-end motion planning. Wang et al. [265] provides a survey article on learning-based motion planning algorithms. Different types of learning-based motion planning approaches include end-to-end neural planners [71, 92, 137, 198, 201, 283], combination of neural network and conventional motion planners [42, 73, 114, 167, 195, 198, 282], and reinforcement learning-based motion planners [17, 62, 112]. These approaches have shown significant improvement in the motion planning capability and runtime for various motion planning problems. As described in the previous section, sampling-based motion planning approaches use sampling of the C-space to find collision-free nodes and edges between the start and end poses to find a collision-free path. The probability of finding a path between two poses depends upon the number of samples in the C-space and the quality of these samples. Thus, a sampling-based motion planning approach's success rate and computational demands depend upon the sampling efficiency. Traditional random sampling-based motion planning approaches often suffer from higher computation and lower success rates for complex environments (cluttered) and/or high-DOF robots. The efficiency of sampling can be improved by performing informed sampling of C-space. More recently, deep learning approaches have been proposed for sampling, as deep neural networks [42, 73, 114, 195, 198, 282].

Motion Planning Network (MPNet): Qureshi et al. [198] proposed a motion planning network for informed sampling in motion planning. Fig. 2.4 represents the proposed neural planning approach. Here, $\hat{x}_t \in \mathbb{R}^d$ represents a sampled pose in the configuration space ($d-$dimensional) of a robot with d degrees of freedom.

x_{start} and x_{goal} represent the start and goal configurations of the robot. The environmental obstacle information x_{obs} (e.g., point cloud) is passed through the Encoder Network. The current step $\hat{x}_t$, goal position x_{goal}, and encoded environment occupancy information E-Net(x_{obs}) are provided to the neural planner, which outputs the next step $\hat{x}_{t+1}$ in the trajectory. The neural planner is iteratively executed until a collision-free path between the start and goal is found. Thus, MPNet provides a trajectory between the start and goal positions of the robot consisting of M milestones $\{x_{start}, \hat{x}_2, ..., \hat{x}_t, \hat{x}_{t+1}, ..., \hat{x}_{M-1}, x_{goal}\}$. This path can also be passed through path smoothing algorithm [198] to find a smooth and near-optimal path. Here, collision detection is used to find the feasibility of planned trajectory and path smoothing. The neural planner is trained using multiple environmental scenarios and generated trajectories $\{x_{start}, x_2, ..., x_t, x_{t+1}, ..., x_{M-1}, x_{goal}\}$ using a conventional planner (e.g., RRT*). A neural planner-based informed sampling helps find a path that is more likely to be collision-free, reducing the planning runtime. MPNet has shown significant improvement in the motion planning success rate and path quality for 2D path planning and high-DOF robot motion planning.

2.1.3 Collision Detection

Collision detection is a crucial task in autonomous robotics and is used to determine the safety of a robot's movements in the environment. Collision detection is the most time and energy-consuming part of motion planning [18, 171, 186, 198]. Collision detection is used in sampling-based motion planning to find if a node (i.e., robot's pose) or edge (robot's short motion between two poses) in the C-space is in C_{free} or not (i.e., collision-free). This collision detection corresponds to an intersection test between the space occupied by a robot's pose or motion and environmental obstacles. Collision detection for a motion can be performed by discretizing the motion into multiple discrete poses, and these poses are checked for collision. Figure 2.5a and Figure 2.5b give examples of collision detection between a robot's pose and motion and environmental obstacles, respectively. We further explain two crucial design aspects of a collision detection algorithm: geometric representation and intersection test. Readers may refer to [58] for more details about collision detection algorithms.

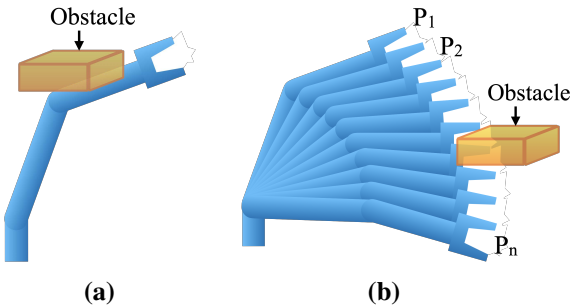
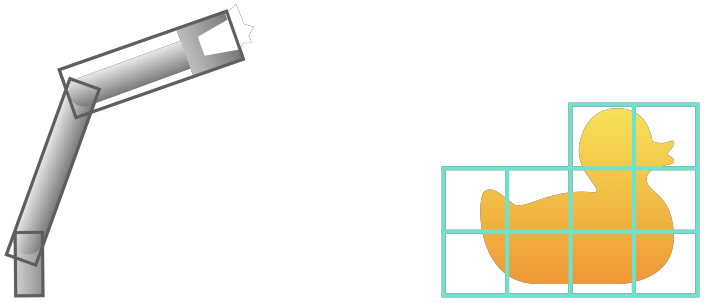


Figure 2.5: (a) represents pose-environment collision detection for a robot, and (b) represents collision detection for a motion from P_1 to P_2, which is discretized into multiple poses to be checked for collision with the environment.

Geometric Representation

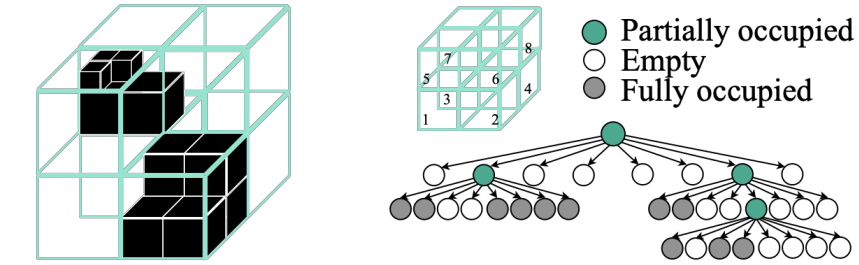
A key design consideration for a collision detection algorithm is the geometric representation of objects. The geometric representation decides the data structures and primitives used to store the space occupied by the robot or obstacles. In the simplest form, an object can be represented as a set of primitives such as spheres, cubes, boxes, or oriented boxes. For example, the environment can be discretized into fixed-size cubes (also known as voxels). Partially or fully occupied voxels are set to 1, and the rest are set to 0. The discretization process and use of primitive shapes result in imprecision in the representation. The shape and size of primitives determine the representation precision and the storage requirement. For collision detection between two sets of primitives, an intersection test between all pairs of primitives is performed until a collision is found or all pairs are collision-free. Figure 2.6a represents the space occupied by a robot using a set of oriented boxes. Similarly, Figure 2.6b represents the space occupied by an environmental object using a set of axis-aligned boxes.

Bounding volume hierarchies have been proposed to reduce collision detection time and storage requirements [133]. In this approach, a tree-type structure represents the space occupied by objects. For example, octree representation can be used to store voxel-based occupancy information. Figure 5.5b shows an example of an octree representation of the occupied voxels from Figure 5.5a in 3D. In octree



(a) **(b)**

Figure 2.6: Geometric representation of a robotic arm (left) and evironmental obstacle (right) using a set of primitive shapes (oriented/axis-aligned rectangles) in 2D.



(a) Spatial division of the occupied space

(b) Octree structure

Figure 2.7: Octree representation (figure adapted from [153]).

representation, the root node represents the entire space in consideration. Each node divides the space into octants and stores the occupancy information (empty/fully occupied/partially occupied) of all octants. Partially occupied octants are further divided into smaller octants. The root node and all partially occupied nodes are stored in memory using a tree data structure. Each node in the tree contains two fields: "`status`" and "`next_addr`". Here, `status` represents the occupancy information of all octants in the current node, and `next_addr` represents a pointer to the memory address storing nodes corresponding to partially occupied octants. Collision detection between an object and the space occupied by this octree is performed by traversing the tree. For each visited node, an intersection test between the object and bounding boxes corresponding to occupied octants is performed.

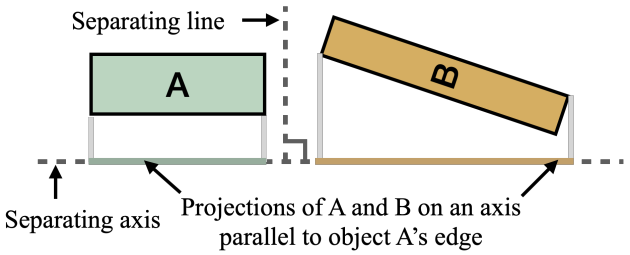


Figure 2.8: Separating axis test to find if two convex objects overlap. Here, a separating axis is found as projections of objects A and B on this axis do not overlap.

The node corresponding to a partially occupied octant is further traversed if an intersection is found with the octant. A collision is found if the intersection test between the object and a fully occupied octant returns true. Octree-based geometric representation for collision detection significantly reduces the number of collision checks, and easy collision-free cases can be tested by traversing only a few nodes in the tree.

Intersection Tests

Intersection tests between two primitive shapes are fundamental to collision detection. The shape of primitives determines the intersection algorithm. For example, two spheres intersect if the sum of their radius is less than the distance between their centers. A more generic intersection approach is the separating-axis test, which can be used to perform an intersection test between any two convex objects [58]. The basic idea behind the separating axis test is that two objects do not overlap if a line or plane exists that separates two 2D or 3D objects. The separating axis is any line perpendicular/normal to the separating line/plane. Figure 2.8 gives an example of a separating-axis test between an axis-aligned bounding box (AABB) A and an oriented bounding box (OBB) B. Here, projections of A and B on the axis parallel to object A's edge do not overlap, which shows that a line perpendicular to this axis separates the two objects. A separating axis is found in this case. For 3D objects, the projection of two objects on an axis orthogonal to a separating plane would not overlap. Depending on the shape of objects, candidates for possible separating axes

can be determined. For example, there are 15 separating axis candidates to perform an intersection test between two 3D OBBs [85], as there are 15 possible separating planes. A plane parallel to a face of either of the OBBs is a candidate for separating planes, which gives six separating axes (three unique face orientations per OBB). Further, a plane spanning the axes of two edges, one from each OBB, is also a candidate for separating planes. The separating axis corresponding to this plane can be found using the cross-product of the axes of these two edges. Since each box has three unique edge directions, there are 3×3 candidates for separating axes. Two objects are determined to be colliding if none of these 15 axes is a separating axis.

2.2 Deep Learning

Deep learning is a class of machine learning methods that use neural networks consisting of more than one layer. Typically, deep learning networks are used for different tasks such as regression, image classification, and language processing. Several methods for deep learning have been explored, such as supervised learning, unsupervised learning, imitation learning, reinforcement learning, and so on. The field of deep learning is vast, with several advancements made in the last decade. This section covers relevant background on deep regression networks and the use of label encodings in deep learning.

2.2.1 Deep Regression Networks

Deep regression networks predict one or more real-valued outputs for a given input. Deep regression networks are used for several tasks, including human pose estimation, age estimation, motion planning, robotic control, scene understanding, neural radiance fields, and occupancy mapping. Several of these tasks are used in robotic perception, planning, and control. Typically, deep regression networks are trained by minimizing the mean squared or absolute error between the predicted and target labels, which is referred to as direct regression. Prior works have also demonstrated advantages of using hybrid loss functions (combination mean squared and absolute error) and regularizers for regression networks.

However, there is a significant gap in accuracy between direct regression and recent task-specialized approaches for regression problems, including head pose es-

timation, age estimation, and facial landmark estimation [31, 264, 276]. Prior works have empirically shown that direct regression results in higher errors compared to specialized approaches. Belagiannis et al. [15] showed that MSE loss used for direct regression is not robust to outliers present in the training dataset and proposed a robust loss function that combines classification loss with regression loss for the same. Several works have shown that the use of direct regression using MSE/MAE loss assumes Gaussian/Laplacian distribution of label uncertainty, which limits the expressiveness of the network [90, 152]. Whereas, Prokudin et al. [194] and Yang et al. [277] showed that for some regression problems, such as head pose estimation, the discontinuity at the boundary (0 and 360 degrees are the same angles but different values) results in a higher error for direct regression. Though task-specialized approaches are effective for limited tasks, given the increasing importance of deep regression networks, developing generic approaches to improving their accuracy is desirable.

2.2.2 Label Encoding in Deep Learning

Label encoding is an approach to using encoded target labels to train a deep neural network. Label encoding has been proposed for multiclass classification problems [4, 45, 54, 229, 261]. Cissé et al. [45] proposed to use an autoencoder to learn label encoding for classification problems with an extremely high number of classes. In this work, the similarity between two classes is used to train an encoder and decoder network such that the similarity between two classes is preserved in learned encodings (output of encoder network). Wan et al. [261] proposed to use Hadamard code to improve the robustness of a classification network. Hadamard code is a widely used error correcting code, where the hamming distance (i.e., number of differing bits) between any two codes of length N is $N/2$. Dietterich and Bakiri [54] demonstrated the advantage of using error-correcting and random codes for label encoding in classification problems. However, these approaches focus on classification problems and do not consider the characteristics of regression problems.

Prior works have proposed binary classification-based approaches using binary label encoding for ordinal regression [44, 47, 149]. Ordinal regression is a class of

supervised learning problems where the samples are labeled by a rank that belongs to an ordinal scale. Ordinal regression approaches can be applied to regression by discretizing the numeric range of the real-valued labels [16, 72]. In the existing works on ordinal regression by binary classification, $N-1$ binary classifiers are used for target labels $\in \{1,2,...,N\}$, where classifier-k predicts if the label is greater than k or not for a given input. This formulation is equivalent to using unary label encoding. Li and Lin [149] provided a reduction framework and generalization bound for the same. However, the proposed application of label encoding and binary classification formulation is restricted. It requires several binary classifiers if the numeric range of output is extensive, whereas reducing the number of classifiers by using fewer quantization levels increases quantization error. Thus, a more generalized approach for using label encoding and binary classification for regression is desirable to allow flexibility in the design of classifiers.

2.3 Soft-error Resilience

In modern systems, errors induced by particle strikes and radiation, or soft errors, make up the majority of SRAM and register-level faults [164, 236]. A soft-error-induced fault in application-specific hardware accelerators in robotics applications can result in erroneous output, leading to a safety violation. For example, the IEC 61508 [115] provides functional safety standards for electronic systems used in applications such as robotics in terms of allowable dangerous failure rate per hour (Section 2.3.2). Hence, soft errors are a major threat to compliance with safety standards in robotics accelerators. This section provides a brief background of functional safety in robotics, soft-error fault characterization, and error mitigation.

2.3.1 Failure-in-Time Rate

The Failure-in-Time (FIT) rate of a circuit (where 1 FIT is one failure/billion hours) consisting of multiple components can be computed using equation 2.1 [147, 170].

$$\text{FIT} = \sum_{i \in \text{components}} \text{S}_i \times \text{SDC}_i \times \text{FIT}_{\text{Raw}} \tag{2.1}$$

FIT_{Raw} is the raw FIT rate defined in FIT/Mb and depends upon multiple factors, including technology node, ambient conditions, and elevation [211]. S_i is the number (in Mb) of sequential elements/latches in component i. Silent Data Corruption (SDC_i) is the probability of a fault in component i affecting the application output.

2.3.2 Functional Safety in Robotics

Safety is a crucial consideration in robotics. Hence, the failure rate of circuits used in robotics applications, including Motion Planning Accelerators (MPAs), is an important factor. IEC 61508 [115] defines an international safety standard for safety-critical electronic systems. This standard is based on the risks of failure and defines four Safety Integrity Levels (SIL). Each SIL expresses the upper bound on the average frequency of dangerous failures per hour (FPH) [162]. SIL 1 is the least stringent, while SIL 4 is the most stringent. The maximum allowable FPH rate decreases by three orders of magnitude from SIL 1 (10^{-5}) to SIL 4 (10^{-8}). Note that the IEC 61508 standard considers the entire electronic system, not only the MPA. More details about functional safety for integrated circuits can be found in [162].

One approach to making circuits soft-error-resilient for certifiable safety is to use hardware error mitigation techniques on storage, which incurs high cost/power/performance overheads. Autonomous vehicles and robotics industries typically have shallow profit margins. For example, the profit margin per unit is under $1000 for several automobile industries [268]. Electronic systems contribute up to 40% to the total cost of a car [51, 203] (at the time of writing). Hence, cost-effective solutions to make MPAs more reliable are imperative [48]. The overheads for complete protection of storage structures increase with their size. In such a case, the protection may be sacrificed entirely if the area/power overheads are over budget. In comparison, selective protection is flexible and provides safety with less overhead than complete protection of storage structures by protecting only the most vulnerable data. Table 2.1 compares the die area and cost for complete and selective ECC for the A3 [153]. As shown in Table 2.1, selective ECC can reduce the cost by 10% for SIL 3 ($\sim$ 1% increase in the profit margin). While the table compares only the die cost, an increase in the die area has a cumulative effect on the total cost of an electronic system, amplifying the need for selective error mitigation.

Table 2.1: Comparison of complete and selective ECC. The die cost calculation is based on the equation provided in [106] (Chapter 1.6). The wafer cost, yield, and impurity factors are for 16nm technology node [34, 53, 239].

	No ECC	Full ECC	Selective ECC (SIL 2)	Selective ECC (SIL 3)
Total area (mm^2)	450	502.5	454.5	469.8
Cost/die ($)	59.9	70.0	60.8	63.6

2.3.3 Fault Injection (FI)

In a circuit, a soft error can occur at any location and time. Assuming a single-bit error model, where only one bit is affected by a soft error in the component, exhaustive fault characterization typically requires A $\times$ B Fault Injection (FI) runs. In this equation, A is the number of fault sites in space, and B is the number of fault sites in time. A is determined by the number of bits. B is determined by the application's total execution cycles and the number of possible inputs to the application. Unfortunately, this requires a high number of FI runs. In MPAs, different combinations of obstacle positions (input to collision detection) add to the number of FI runs, making the space even larger.

Accelerators' fault characterization is typically carried out by statistical FI experiments [37, 68, 97, 98, 145, 147]. Statistical FI performs random sampling in the fault space and allows tuning the number of FI experiments as per the required confidence of the estimated failure rate for a system [145]. The number of FI runs needed for statistical FI increases with a decrease in the probability of a bit error resulting in an output error, i.e., the failure probability of a component [145]. Often, statistical FI is performed per component to find the safety-critical components in a circuit. Statistical FI grouping needs to be carefully designed to find safety-critical components in a given circuit.

Chapter 3

Motion Planning Acceleration

This chapter describes our contributions to robot motion planning acceleration. Motion planning is a crucial task in autonomous robotics that aims find a collision-free and feasible path for a robot to its goal position in the physical environment. Section 2.1 provides background related to robot motion planning and its challenges. As mentioned before, motion planning is a computationally intensive task with demands that increase rapidly with the number of degrees of freedom (DOF) of the robot and environment complexity [217]. Thus, improving the performance and energy efficiency of motion planning is important to enabling the deployment of robots for challenging environments and tasks. Robots with higher degrees of freedom (e.g., the 7-DOF Baxter robotic manipulator [208]) can perform more than one task or complex tasks in a cluttered environment. While the cost of high-DOF robots is decreasing, the latency of motion planning for high-DOF autonomous robots is currently a major impediment to its deployment [232, 244, 247].

Several acceleration approaches have been proposed for collision detection and traditional motion planning algorithms to meet the real-time computation requirements, including on GPUs [18, 76], FPGAs [8, 171, 226], and ASICs [11, 122, 153, 173, 233, 278]. Bakhshalipour et al. [11] proposed a voxelized robot-environment collision detection approach. Bakhshalipour et al. [11] also introduced RACOD and proposed a speculative parallelism-based accelerator for path planning of robots with 2 or 3 DOFs (e.g., autonomous cars or drones). However, the proposed approach does not apply to motion planning algorithms for robots with higher

DOFs [11]. Other works have proposed hardware accelerators for sampling-based motion planning algorithms suitable for high-DOF robots [153, 171, 173, 233, 278]. The reason is that the underlying motion planning algorithm (i.e., probabilistic roadmaps) used in these works requires significantly more computation as the complexity of robotic tasks and environment increases. For example, a probabilistic roadmaps-based motion planning accelerator suitable for dynamic environment and challenging tasks requires more than 40MB on-chip memory or 40GBPS off-chip memory bandwidth [153, 173].

There have been improvements in the field of motion planning algorithms. More recently, informed sampling-based motion planning algorithms have exhibited significant improvement in the computation requirement, path quality, and success rate of motion planning compared to conventional algorithms [113, 142, 199, 200, 265, 282]. These approaches use different heuristics to improve the sampling efficiency and reduce collision checks. For example, learning-based motion planning approaches use neural networks for sampling [113, 142, 199, 200]. MPNet [200], one of the recent learning-based motion planning approaches, has shown 15× speedup on CPU and 40% improvement in the path quality compared to the traditional sampling-based motion planning algorithms. Though such more efficient algorithms have been explored and studied at the software level, their architectural implications have not been studied.

For sampling-based motion planning algorithms, collision detection between the robot and the environment consumes $\sim 90\%$ of execution time [18, 171]. Sampling-based motion planning provides an approximate trajectory for the robot by finding a set of intermediate positions that the robot can take to reach the end goal. For example, a neural network provides this approximate trajectory in learning-based motion planning. The motion between two intermediate positions is generated by a local planner. Typically, the linear interpolation between two positions is used as a local planning approach [140, 197]. Collision detection is used to find which intermediate positions and motions between intermediate positions are collision-free to assess the feasibility of this trajectory. Here, short motions between intermediate positions can be checked for collision in parallel. Similarly, a short motion is discretized into several robot positions, and collision detection for each position is performed, providing inter-collision detection parallelism. Further, for

each collision detection query, different parts of the robot and environment can be checked in parallel for a collision, and the parallelism available in the collision detection algorithm can be used, providing fine-grained parallelism.

There is ample coarse- and fine-grained parallelism in sampling-based motion planning, that can be exploited for acceleration. However, the goal of collision detection in sampling-based motion planning is to find if a given short motion or position of the robot is collision-free. Thus, a collision detection query can be terminated early if a collision is found. For example, a collision detection query between a robot's motion and environment can be divided into multiple intersection tests between different poses of the robot and the environment. For a sequential execution of these intersection tests, subsequent queries can be discarded once an intersection test returns true for collision check. Further, if it were known which poses of the robot are more likely to be in a collision, this "colliding intersection test" could be scheduled first to reduce the number of intersection tests performed to find a collision. For a collision detection query with positive outcome (collision "True"), we call all intersection tests redundant that return collision detection "False", as ideally, it is possible to perform a single intersection test with positive outcome to find the output of this collision detection query.

We find that a significant fraction of computation performed during motion planning collision detection is redundant, and the amount of redundant work performed increases with the degree of parallelization for a naive parallel implementation. Figure 3.1 compares the speedup and computation for sequential and parallel evaluation on specialized hardware. Parallel evaluation results in 50$\times$ speedup with 3.4$\times$ computation compared to sequential evaluation. Thus, naive parallelization of collision detection is work-inefficient, which significantly increases computation and energy consumption. A parallel algorithm is said to be work-efficient if the amount of work done by it is asymptotically equal to the work performed by the fastest sequential algorithm for the same problem and several works have focused on this problem for different algorithms [20, 191, 228].

In this work, we analyze the sources of redundant computation in coarse- and fine-grained parallelization of collision detection and propose an algorithm-hardware optimization approaches to improve the energy efficiency and execution time of motion planning. This chapter describes three major contributions made in this

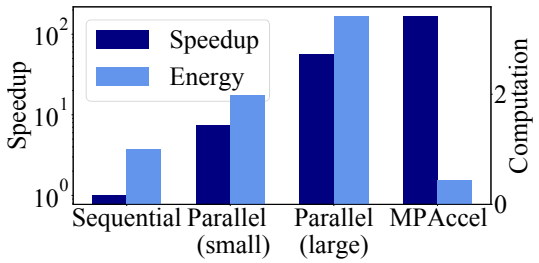


Figure 3.1: Comparison of the speedup and energy efficiency for different execution modes on ASIC hardware. Small and large represent the scale of parallelization. Experimental methodology is described in Section 3.6.

work: Spatially Aware Scheduler (SAS), Cascaded Early-exit Collision Detection Unit (CECDU), Collision Prediction Unit (COPU). Each contribution represents an approach to target a different source of redundant computation in motion planning and a corresponding hardware accelerator. The key insight behind these approaches is that the physical spatial locality in the space occupied by different positions of the robot and environmental obstacles can be exploited to reduce redundant computation in motion planning acceleration.

SAS groups intersection tests for spatially distant positions of a robot in a batch for parallel execution to improve the work efficiency of parallel execution. Collision detection outcomes for spatially nearby robot positions are likely to be similar due to the physical locality of obstacles. Thus, scheduling distant positions in a batch to cover more space is crucial to reduce redundant computation.

A motion-environment or pose-environment collision detection query consists of several intersection tests. We show that the history of outcomes of previous intersection tests can be used to predict the outcome of intersection tests for spatially nearby positions of the robot. Based on this insight, a collision prediction unit (COPU) is used to find potentially colliding tests from a pool of intersection tests to find a collision in fewer tests and reduce redundant intersection tests.

We further show that easy intersection test cases with significantly far or overlapping objects contribute the most to redundant computations in fine-grained parallelism. We propose a cascaded early-exit unit, CECDU, that filters such easy collision-free and colliding cases by performing low-compute intersection tests

using simple geometric primitives (e.g., spheres) bounding and inscribing an object (e.g., robot's link). CECDU performs a precise intersection test only if required.

Finally, we build MPAccel, a motion planning hardware accelerator consisting of SAS and CECDU, and evaluate the runtime and energy consumption of a learning-based motion planning algorithm for a 7-DOF robot on MPAccel.

For $8\times$ parallelization, SAS results in a $7\times$ speedup with a 6% increase in the computation compared to a $3.7\times$ speedup with an 83% increase in computation for a naive scheduling. COPU results in $17.2\% - 32.1\%$ reduction in the collision detection queries on average across different motion planning algorithm-robot combinations compared to spatially aware scheduling policy. This computation reduction reduces the energy consumption of collision detection by 32% for MPNet motion planning algorithm [198]. CECDU can perform collision detection in $46 - 154$ cycles for a 6-DOF robot. MPAccel, consisting of the proposed SAS and CECDUs, enables realtime motion planning for a 7-DOF robot using a learning-based motion planning algorithm in 0.014ms-0.49ms with 0.099ms on average, and consumes 3.5W power for 45nm technology node. In summary, we make the following contributions in this chapter:

- We study the sources of redundant computation in coarse-grained and fine-grained parallelism in motion planning.
- We propose MPAccel; it consists of a Spatially Aware Scheduler (SAS) to handle coarse-grained parallelism and Cascaded Early-exit Collision Detection Units (CECDUs) to handle fine-grained parallelism.
- We propose a Collision Prediction Unit, COPU, and integrate it with a collision detection unit. We evaluate the proposed collision prediction unit on various robots, environments, and motion planning algorithms.
- We evaluate a learning-based motion planning algorithm, MPNet, on the proposed MPAccel.

Section 3.1 provides relevant background on collision detection in motion planning approaches studied in this chapter. This chapter covers the details of proposed

approach and microarchitecture for Spatially Aware Scheduler (Section 3.2), Collision Prediction Unit (Section 3.4), and Cascaded Early-Exit Collision Detection Unit (Section 3.3). Further, it introduces and provides details of MPACCEL, a realtime energy-efficient hardware accelerator consisting of the above blocks (Section 3.5). Section 3.6 and Section 3.7 provide the methodology and evaluation of SAS, CECDU, COPU, and MPAccel. Finally, Section 3.8 discusses the related work, and Section 3.9 provides a conclusion.

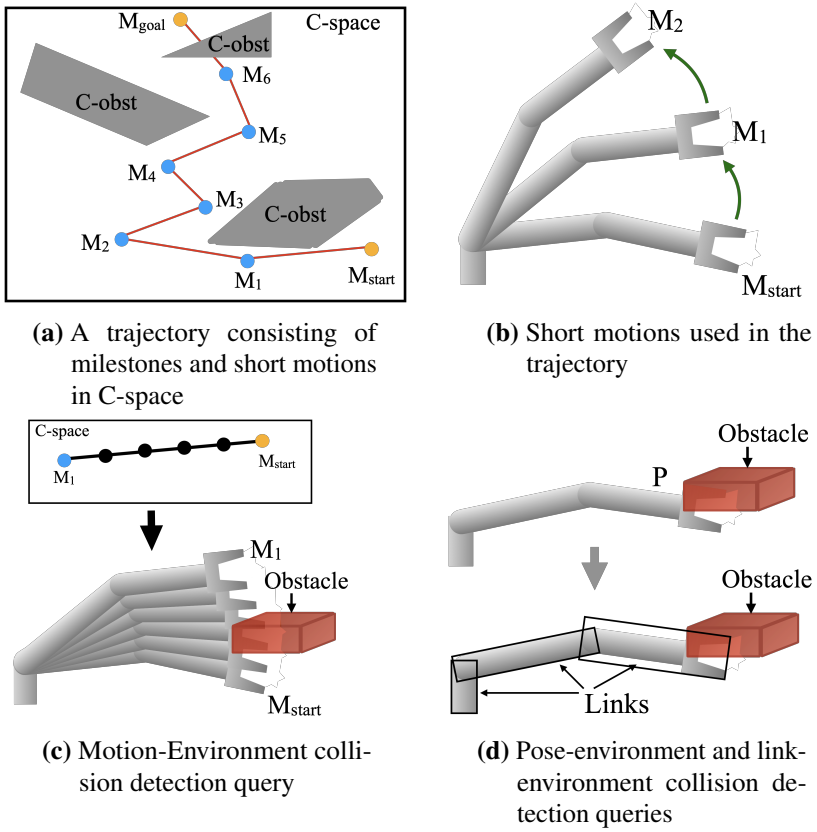
3.1 Background: Collision Detection for Motion Planning

This section briefly summarizes the relevant background of collision detection computation in sampling-based motion planning and defines terms used in the rest of this chapter.

As described in Section 2.1, in sampling-based motion planning, the C-space of a robot is sampled coarsely to find feasible positions and motions for a given robot and environmental scenario. Different types of collision detection queries are performed depending on the motion planning algorithm and sampling strategy. Further, collision detection queries exhibit different types of parallelism.

We take an example of the "feasibility checking" phase (used in MPNet [198] and LazyPRM [21]) to further explain different abstraction levels of collision detection. In this phase, a sampled trajectory consisting of some milestones between the start and end position is to be checked for collision. Figure 3.2a represents a trajectory consisting of milestones $\{M_{start}, M_1, M_2, ..., M_{goal}\}$ in the C-space. Each milestone corresponds to a distinct position of the robot in the physical space. Here, two adjacent milestones are connected by a straight line, representing a short motion between corresponding positions of the robot (M_{start}-M_1 and M_1-M_2 in Figure 3.2b). Here, any deterministic local planner can be used. For simplicity, we assume a local planner that generates linearly interpolated motion between two positions. Figure 3.2c represents such linearly interpolated short motion M_{start}-M_1 in the physical space.

All such short motions are checked for collision with environmental obstacles for collision detection of this trajectory. Each individual collision detection query is termed as *motion-environment collision detection*. Further, each motion



(a) A trajectory consisting of milestones and short motions in C-space

(b) Short motions used in the trajectory

(c) Motion-Environment collision detection query

(d) Pose-environment and link-environment collision detection queries

Figure 3.2

is discretized into multiple discrete poses for collision detection, as shown in Figure 3.2c. Each of these discrete poses of the robot is then checked for collision with environmental obstacles, termed as *pose-environment collision detection query*. Figure 3.2d (top) gives an example of a pose-environment collision detection query. A pose-environment collision detection query is referred to as collision detection query unless specified. Further, depending on the shape and topology of the robot, a robot can be represented using multiple primitives. In this work, we represent each robot link using an oriented bounding box. Thus, a pose-environment collision detection query can be divided into multiple *link-environment collision detection* queries, as shown in Figure 3.2d (Bottom).

In this formulation, parallelism is available at different granularity, and multiple

motions within a given trajectory, poses within a motion, and links within a pose can be checked in parallel in the "feasibility check" phase of motion planning. We refer to parallelism available between multiple motion-environment collision detection queries as *inter-motion* parallelism. Similarly, we refer to parallelism available between multiple pose-environment collision detection queries for a given motion as *intra-motion* parallelism.

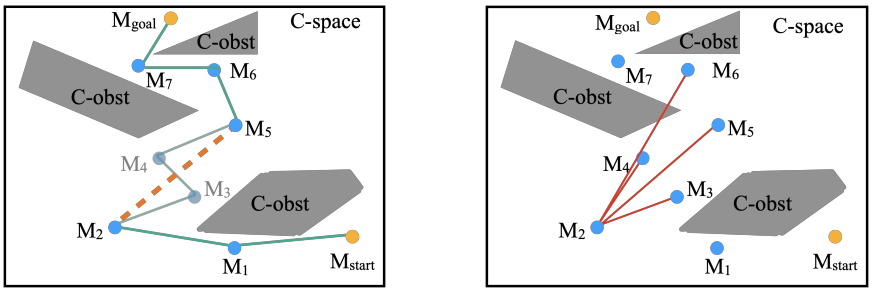
Another example of collision detection computation is for "path optimization" in motion planning. Figure 3.3 gives an example of path optimization phase. Here, a greedy shortcutting algorithm is used to smoothen the trajectory by removing redundant intermediate poses/nodes [70, 77, 101, 200, 220, 281]. In this method, for a given node M_i and end node M_N, multiple edges are checked for collision to find redundant nodes between these two nodes. If a motion between from M_i to M_j is collision-free, poses $M_{i+1}, M_{i+2}, ..., M_{j-1}$ are considered redundant as M_i and M_j are connected. Removal of such redundant intermediate poses results in a smoother path. The goal is to find the highest value of j such that $M_i - M_j$ straight motion is collision-free. In the example given in Figure 3.3, $M_2 - M_{goal}$, $M_2 - M_{goal-1}$, ..., $M_2 - M_3$ motions are checked for collision in this order to find redundant nodes between M_2 and M_goal. However, there is no data dependency between these motion-environment collision detection queries, and these can be executed in parallel at the expense of higher computation. Thus, this phase of motion planning consists of inter-motion and intra-motion collision detection parallelism.

SAS targets parallelism at the level of motion-environment and pose-environment. CECDU focuses on the acceleration of single pose-environment collision detection query. COPU focuses on parallelism available at the level of pose-environment and link-environment.

3.2 Spatially Aware Scheduler (SAS)

This section details our analysis of coarse-grained parallelism (i.e., inter-pose-environment collision detection queries) in motion planning. Further, the proposed approach to exploit this parallelism in an energy-efficient manner, Spatially Aware Scheduler (SAS), is explained in detail.

Each phase of motion planning provides coarse-grained intra-motion parallelism



(a) Redundant nodes in a path

(b) Collision check for path optimization using shortcutting

Figure 3.3: (a) Example of redundant nodes in a path found between start and goal. Here, nodes M_3 and M_4 are redundant as the direct path between M_2 and M_5 is collision-free. (b) Example of path optimization using shortcutting in motion planning. Here, multiple edges are checked for collision to remove redundant nodes between M_2 and M_{goal}.

(Section 3.1). We first perform a limit study to analyze the impact of the number of pose-environment collision detection units (CDUs) (i.e., degree of parallelization) on the number of Collision Detection Queries (CDQs) performed during motion planning (i.e., a measure of work efficiency) and collision detection runtime. Benchmarks used for this study are described in Section 3.6. The limit study assumes zero cycle latency for the scheduler and latency of one cycle for a CDQ. We observe that the number of CDQs performed increases by 2.4× with 12.4× reduction in the runtime for 16× naive parallelization. As the degree of parallelization increases, the number of collision tests and energy increase with speedup.

The key reason behind this mismatch in the number of collision detection runs between sequential and parallel evaluations is that once a collision is detected for any robot pose along a motion, there is no need to perform collision detection for the following poses from this motion. Figure 3.4a represents a robot's pose in the physical space and C-space. Black and red dots represent the discrete poses checked for collision detection of this motion. Figure 3.4b.i represents the sequential evaluation of collision detection for a motion. A collision is detected and completed for this motion at cycle 5. However, parallel evaluation using 4 CDUs, as shown in Figure 3.4b.ii, results in more collision detection queries (executed)

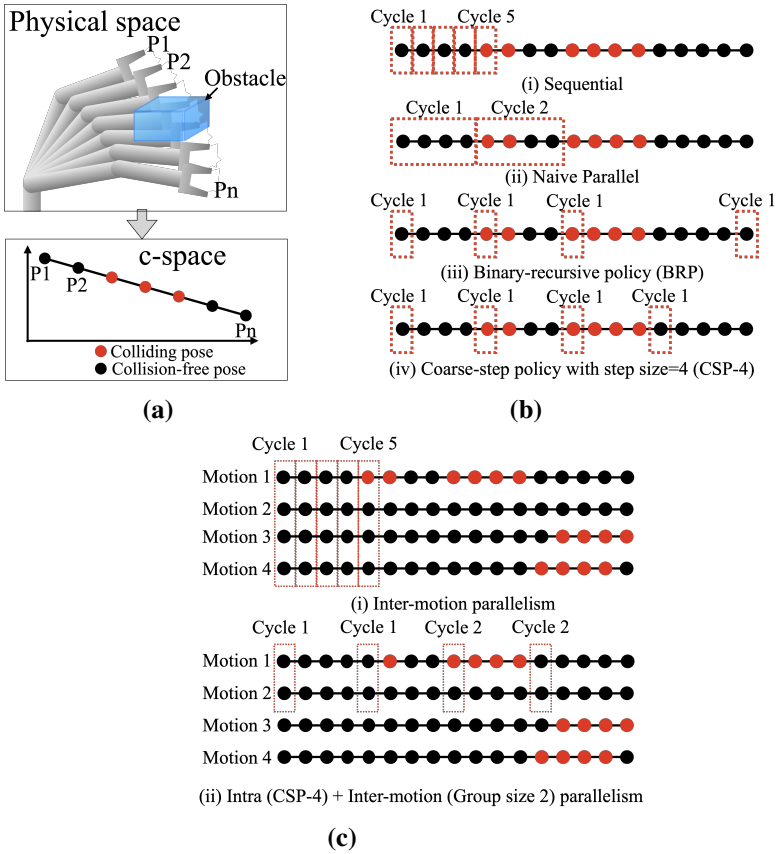


Figure 3.4: (a) represents a 2-DOF robot's motion in the physical and C-space. Here, a dot in the C-space represents the robot's pose in discretized motion. Collision detection for all discrete poses is performed for the motion's collision detection. (b)-(c) represents different scheduling approaches for sequential and parallel evaluation (4 CDUs) of collision detection queries. (b.ii)-(b.iv) are examples of intra-motion parallelism, (c.i) represents only inter-motion parallelism (referred to as multi-motion), and (c.ii) represents an example of intra+inter motion parallelism.

compared to sequential evaluation. Thus, naive parallel execution provides speedup at the expense of executing more collision detection queries. This effect becomes more pronounced as the scale of parallelization increases, as shown in Figure 3.5 (NP). Another approach is to use inter-motion parallelism. However, different motions are not necessarily independent tasks in motion planning. For example,

in shortcutting for motion planning (explained in Section 3.1), the goal is to find the first collision-free motion from a pool of motions [77, 101, 220, 281]. In Figure 3.4c.ii, collision-detection for motion 3 and 4 is redundant as motion-2 is collision-free. Inter-motion parallelism reduces redundant computation compared to naive parallelism. However, its effectiveness reduces as the scale of parallelization increases (See MS in Figure 3.5). Thus a combination of intra- and inter-motion parallelization is required.

The increase in redundant computation for intra-motion parallelism is due to the physical spatial locality of the robot's poses and obstacles. There is considerable overlap between the physical space covered by the poses of the robot corresponding to adjacent points, as shown in Figure 3.4a. Hence, in most cases, collision detection results for nearby poses (i.e., nearby points in Figure 3.4) are likely to have the same output. Naively grouping adjacent poses in a batch for parallelization degrades work efficiency, as spatially similar poses are checked together. A remedy to this inefficiency is to schedule physically distant poses in a batch for parallel evaluation to cover more space. Based on this insight, we propose a Spatially Aware Scheduler (SAS), which schedules physically distant poses in a batch. We explore two scheduling policies for SAS.

We first explore a binary-recursive traversal-based scheduling policy. The difference between the indices of poses gives a measure of the physical distance between the poses in Figure 3.4a. Binary recursive scheduling policy (BRP) selects the order of poses using the binary-recursive algorithm, thus sampling the motion from coarse to fine. Figure 3.4b.iii represents scheduling using the binary-recursive algorithm, where poses with identifiers 0, N, $N/2$, and $N/4$ are selected in the first cycle. However, BRP requires maintaining a queue. We also explore a simpler scheduling strategy based on coarse steps. This coarse-step scheduling policy (CSP) uses a value greater than one as the step size to select poses in a batch. For example, for a step size of four, points 1 to N are scheduled in the order of $\mathbf{0}, 4, 8, ..., \mathbf{1}, 5, 9, ..., \mathbf{2}, 6, 10, ..., \mathbf{3}, 7, 11, ..., N$. Figure 3.4b.iv represents CSP for a step size of 4. CSP schedules discrete poses in a motion in a coarse-to-fine manner to avoid *physical-locality-induced redundant collision checks*.

We combine inter-motion and intra-motion parallelism to reduce redundant computation in the parallel execution of collision detection queries. We propose

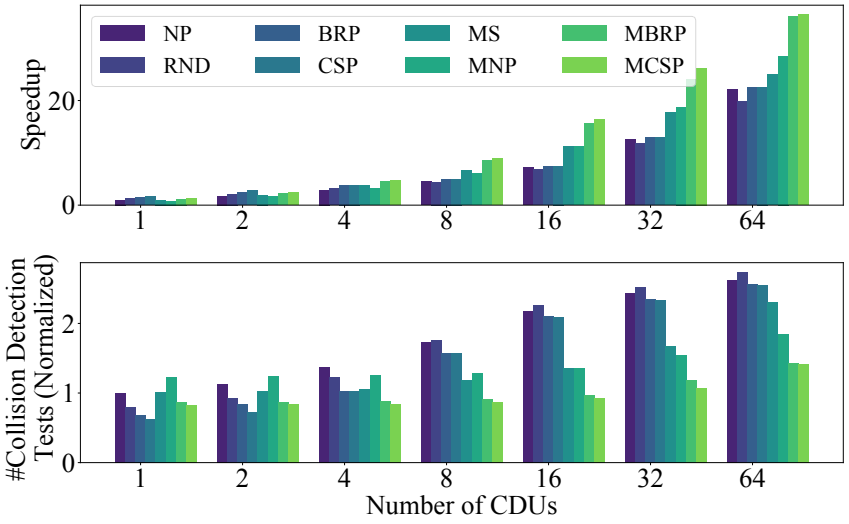


Figure 3.5: Limit study on the effect of scheduling policies on the number of collision detection cycles and runs for different numbers of CDUs. NP: Naive parallel, RND: Random scheduling; BRP: Binary recursive policy, CSP: Coarse-step policy; prefix M represents inter-motion parallelism. MS represents only inter-motion parallelism.

Multi-Motion Coarse-Step Scheduling Policy (MCSP) for SAS that combines CSP with inter-motion parallelism to take advantage of both kinds of parallelism. In MCSP, a group of motion, determined by the group size for inter-motion parallelism, is considered for scheduling. Within a motion, the order of poses is selected based on CSP. Figure 3.4c.ii represents the MCSP approach for the group size of two and step size of four.

Figure 3.5 represents a limit study on the number of collision detection queries and runtime for different scheduling approaches. We also compare with a random selection of points within a motion. We compare different combinations of {Seq (S), Naive Parallel (NP), Random (RND), Coarse-step Policy (CSP), Binary Recursive Policy (BRP)} and {With inter-motion parallelization (M), Without inter-motion parallelization (M omitted)}. For inter-motion parallelization, selected poses in a single batch can be from more than one short motion. As shown in the figure, naive or naive+inter-motion parallelization is not sufficient for energy efficiency

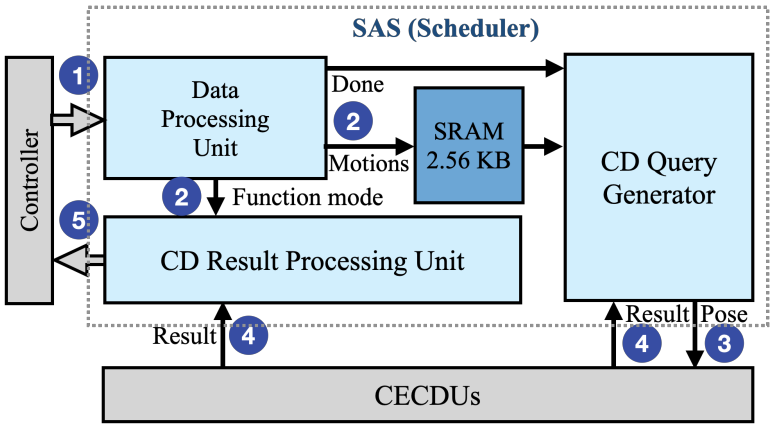


Figure 3.6: SAS microarchitecture.

and speedup for a higher number of CDUs. Furthermore, CSP results in faster collision detection than the ordered selection of poses for sequential evaluation (i.e., #CDU=1) because of the efficient exploration of the space covered by a motion. We also see that CSP performs very similarly to the BRP and translates to a simpler hardware/software implementation as the binary recursive approach requires maintaining one or more hardware/software queues. In contrast, coarse-step-based scheduling can be implemented using registers and adders. *The figure shows that MCSP can achieve up to* 13.5× *speedup using* 16 *CDUs with only a* 10.5% *increase in the number of collision detection tests.*

3.2.1 Microarchitecture

This section explains the microarchitecture of the hardware implementation of the proposed spatially aware scheduler. Figure 3.6 represents the microarchitecture of SAS. SAS receives a group of one or more motions for collision detection from the controller. SAS supports three types of function modes. The "Feasibility test" mode is used to find if all motions in a group are collision-free. In this case, the scheduler stops once a collision for any pose is found. The "Connectivity test" mode is used to find if at least one motion is collision-free. In this case, the scheduler stops when one collision-free motion is found and all motions proceeding it are found to be colliding. This functional mode is useful for path optimization (Section 3.1). The

"Complete test" mode is to get collision detection results for all motions.

The Data Processing Unit receives the data sent by the controller (1), consisting of metadata and motion data for a group of motions. The metadata includes the number of motions and the function mode. Motion data contains its start pose, the distance between two discrete poses, and the number of discrete poses. The Data Processing Unit processes and sends the received data to other units and SRAM (2). The Collision detection (CD) Query Generator implements the logic to order the poses for collision detection as per the MCSP. The CD Query Generator generates discrete poses to be checked for collision and sends it to free CECDUs (3) in the order determined by MCSP. The step size for MCSP is set to 8. Similarly, group size, i.e., the number of motions considered for inter-motion parallelism in MCSP, is set to 16. We determine the step size and the group size for MCSP based on empirical evaluation for the MPNet motion planning algorithm using a subset of benchmarks.

The CD Query Generator also receives the collision detection results from the CECDUs (4). It removes a motion from the scheduling list if an intermediate pose for this motion is found to be colliding. This way, it ensures not to schedule redundant work to the CECDUs. The CD Result Processing Unit receives collision detection results for all queries from the CECDUs (4), and maintains collision detection output for each motion in the current group. Here collision detection output for a motion is stored using single bit (True or False). Depending upon the function mode, the result processing unit signals other units to stop operation and sends the result (collision output of all motions) to the controller (5).

3.3 Cascaded Early-exit Collision Unit (CECDU)

Collision detection is a widely studied problem with applications in various fields. In motion planning, collision detection is used to find if the robot collides with its surroundings for a given pose or motion. This section summarizes the intra-collision detection query parallelism analysis. Further, the proposed Cascaded Early-exit Collision Detection Unit, CECDU, is explained in detail.

3.3.1 Geometric Representation and Intersection Test

An important aspect of collision detection algorithm and its acceleration is the geometric representation used for the robot and the environment. Section 2.1.3 provides a background on geometric representation and intersection tests for collision detection. We consider mainly three factors for selecting the geometric representation and intersection test algorithm. The first factor we consider is the calculation of the robot's occupied space for a pose. Prior works on collision detection acceleration have proposed to precompute the robot's occupied space for different poses or motions and store it in memory [153, 172, 173, 278]. The advantage of such a method is that the representation can be optimized offline for storage and computing efficiency both. However, such precomputation does not allow collision detection for arbitrary poses explored by a motion planning algorithms. Furthermore, the storage requirement increases with the complexity of the robot and its tasks [153, 173]. We consider on-chip calculation of the robot's occupied space for a given pose. Thus we rule out the use of bounding volume hierarchy (BVH) for the robot's geometric representation, as BVH tree generation is compute-intensive [58]. Based on this insight, we use a set of oriented bounding boxes (OBB) to represent the robot. As the robot changes its pose, each link goes through a rigid transformation, i.e., its orientation and translation change. The size of the bounding box for each link of the robot can be precomputed. At runtime, the robot's pose (e.g., angle of its joint) is used to find the orientation and center of these OBBs using trigonometric functions and matrix multiplication [52].

The second factor is the collision detection computation requirement for colliding and collision-free cases. We find that more than 95% of the OBB-environment collision tests are collision-free in motion planning benchmarks used in this work (Section 3.6). BVH tree-based representation reduces computation for collision-free cases, as collision detection can terminate if no collision is found at a node. As mentioned earlier, generating a BVH tree for representing the space occupied by a robot's pose is expensive as more than 1000 poses are tested for collision for each motion planning query. However, the environment is updated only once for a motion planning query. Based on this observation, we use an octree representation of the environment (Section 2.1.3). Prior works have focused on mapping sensor

data (e.g., point cloud, 2D images) to octree [107, 122, 245]. Jia et al. [122] proposed a mapping accelerator to build octree from point cloud data. Such mapping accelerators can be used to provide the environment's octree representation.

The third factor we consider is the scalability of the intersection test for higher precision or a larger environment. Collision detection between a robot (represented by a set of OBBs) and environment (represented by an octree) consists of multiple intersection tests between OBBs and AABBs (from the octree nodes). One simple approach includes rasterization of OBB to a set of voxels. Collision detection is performed between voxels and environment octree. However, the number of voxels increases significantly with the resolution of rasterization. For example, we find that the number of intersection tests increases by $\sim 5\times$ when the discretization step size is decreased by half for OBBs of the Jaco2 robotic arm [132]. Moreover, this requires checking all voxels for collision-free cases. Considering this, we select the separating axis test for OBB-AABB intersection test. The separating axis test consists of multiplications and additions. The separating axis test allows an accurate intersection test between the OBB and the environment, reducing false positives (i.e., a collision-free pose is flagged as colliding).

3.3.2 Proposed Approach for OBB-Octree Intersection Test

We represent the space occupied by a robot's pose using a set of OBBs and the space occupied by environmental obstacles using an octree. Thus, a pose-environment collision detection test consists of multiple OBB (e.g., link)-octree collision detection tests. Collision detection between an OBB and octree is performed by traversing the octree and performing an intersection test between the bounding box (i.e., AABB) corresponding to a node's octant and OBB (Section 2.1.3). Thus, an OBB-octree collision detection test consists of one or more OBB-AABB intersection tests. In this section, we focus on analyzing the parallelism available within an OBB-AABB intersection test and accelerating it.

As described in Section 2.1.3, the separating axis test provides an exact intersection test between two convex objects. For any object pair, a limited number of potential separating axes exist. Two objects are determined to intersect if none of the potential separating axes is a separating axis. For the OBB-AABB intersection

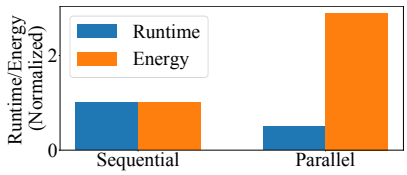


Figure 3.7: Comparison of runtime (#cycles) and energy for sequential and parallel execution of the separating axis tests.

test, 15 potential separating axes exist ((Ch. 4.4.1 in [58])). We can conclude that a given pair of OBB and AABB intersects if none of the 15 potential axes is a separating axis. All 15 tests can be performed in parallel to accelerate the intersection test. Figure 3.7 shows the number of multiplications performed (i.e., approximated energy) for sequential and parallel execution of separating axis tests for collision-free cases. Parallel execution results in approximately $3\times$ increase in the energy. We find that the primary sources of this increase are collision-free cases, where a separating axis is found after testing the n^{th} separating axis candidate, and executing all 15 tests is redundant. To understand the inefficiency of parallel evaluation, we profile the distribution of the number of separating axis tests performed for collision-free and colliding cases in Figure 3.8. We use collision detection tests between OBBs for random poses of Jaco2 robot [132] and octree for random environmental scenarios. In the OBB-AABB separating axis test, the last 9 separating axes consider edge-to-edge placement of two objects, which occurs with low probability [254]. We observe similar trends for our benchmarks, and in most collision-free cases, a separating axis is found in the first six axes. Based on this, we propose a three-stage execution mode, in which the 15 tests are divided as $6-5-4$ among three stages. A later stage is only executed if the previous stage returns false, i.e., a separating axis is not found. The specific division of separating axes tests across different cycles is done to balance the required hardware resources (e.g., multipliers and adders) across different stages for multicycle hardware implementation of the proposed execution model. This modification decreases multiplication operations by $1.5\times$ compared to fully parallel execution.

Furthermore, as shown in Figure 3.8, for most collision-free cases, the first axis returns true for the separating axis test. We find that in most cases where the objects

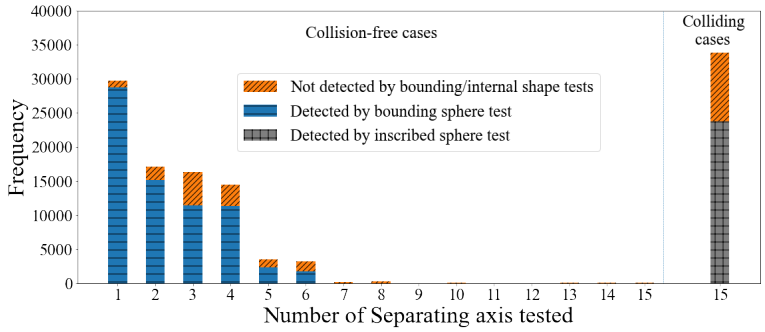
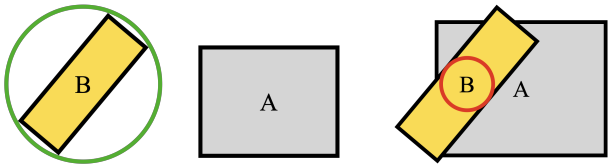


Figure 3.8: Distribution of the number of separating axis tests performed for OBB-octree collision detection.



(a) Bounding sphere **(b)** Inscribed sphere

Figure 3.9: Use of spheres to filter easy cases and reduce computation, where objects are far apart (a) or significantly overlapping (b). Objects are represented in 2D for clarity.

are far apart, a separating axis is found in the first few separating axis candidates. Prior works have proposed to use a computationally simple intersection test between bounding spheres of OBBs before performing a detailed intersection test [38]. Figure 3.9a gives an example of the bounding sphere for an OBB. The intersection test between a sphere and an AABB requires three multiplications compared to 81 for checking possible 15 separating axes for the OBB-AABB intersection test. The blue bars in Figure 3.8 show the fraction of cases filtered by a sphere-AABB test. The majority of the intersection tests that find a separating axis in the first test and hurt the energy efficiency of parallel execution can be filtered by the bounding sphere-AABB test.

Further, we find that after applying the bounding sphere-based filter, $\sim 80\%$ of the operations are used by colliding cases for OBB-AABB intersection tests. Note that for colliding cases, all 15 separating axis candidates are checked before

concluding that there is a collision. We further propose performing an intersection test between an AABB and a simpler shape inscribed in an OBB before an expensive OBB-AABB intersection test. The higher the space covered by the inscribed shape, the more collision cases can be found using simpler intersection tests. For a given pair of AABB-OBB, we use an inscribed sphere for the OBB and check it for intersection with the AABB. An inscribed sphere is the largest sphere inside a shape that touches its edges (Figure 3.9b). Overall, we observe that inscribed sphere-AABB tests find 70.6% of the colliding cases using fewer operations for OBB-AABB intersection tests corresponding to OBB-octree collision tests (Methodology explained in Section 3.6). Further, we find that the inscribed sphere-based test is more effective for AABBs corresponding octree nodes closer to the root node. Our evaluation shows that the proposed inscribed sphere-AABB can find 81% and 40% of the colliding cases for AABBs corresponding to the first and the fourth level of octrees, respectively. For the octree-based representation of an environmental scenario, AABBs corresponding levels closer to the root node are larger compared to AABBs corresponding levels closer to the children nodes. If such large AABBs intersect with considerably smaller OBBs, it is likely that the OBB significantly or entirely overlaps with the AABB. The inscribed sphere-AABB test efficiently finds such colliding cases with fewer operations.

Figure 3.10 represents the flowchart for the proposed intersection test for the proposed Cascaded Early-exit Collision Detection Unit (CECDU). The intersection tests are performed in a cascaded manner, exiting early if collision detection output is found. The function returns collision if a separating axis is not found after checking all 15 axes.

3.3.3 CECDU Microarchitecture

The CECDU receives the robot's pose from the scheduler and performs collision detection between the robot and the environment. Figure 3.11 represents the CECDU microarchitecture. The OBB Generation Unit generates a set of OBBs representing the robot's occupied space for the given pose. The generated OBBs are sent to the OBB-octree Collision Detectors (OOCDs). Each OOCD performs collision detection between an OBB and the environment octree. The Result Collector receives results

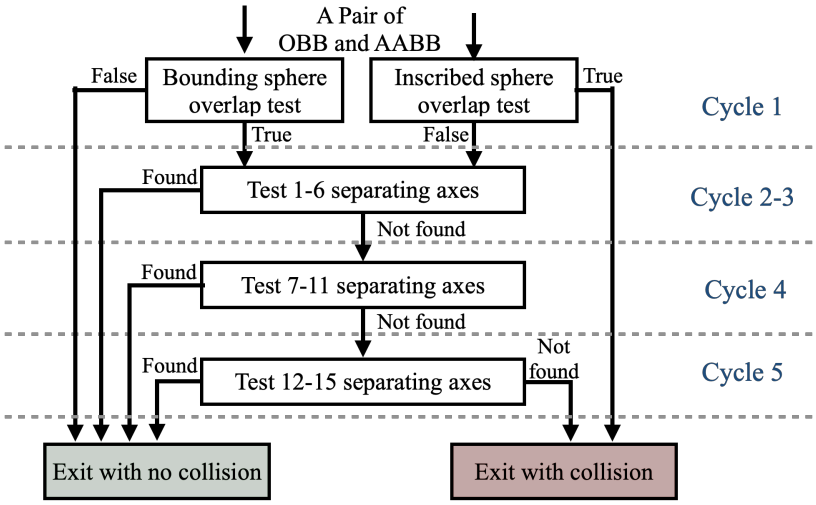


Figure 3.10: The flowchart for the proposed cascaded early-exit intersection test using bounding and inscribed-sphere filters and separating axis test.

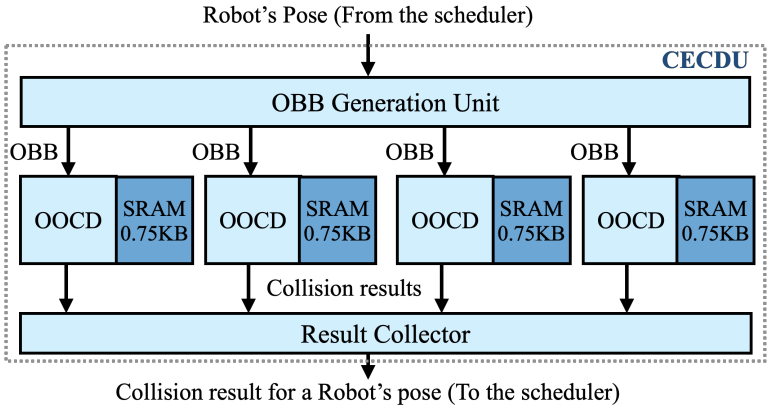
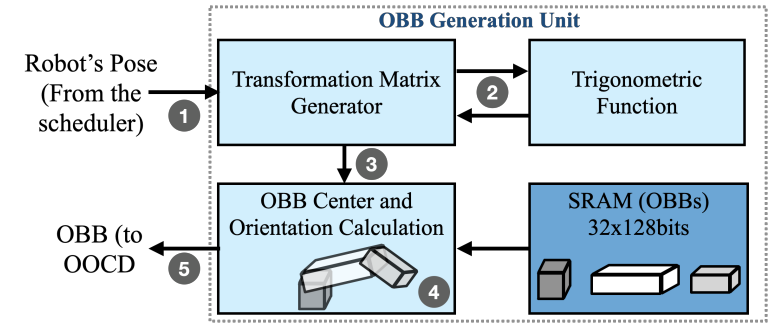
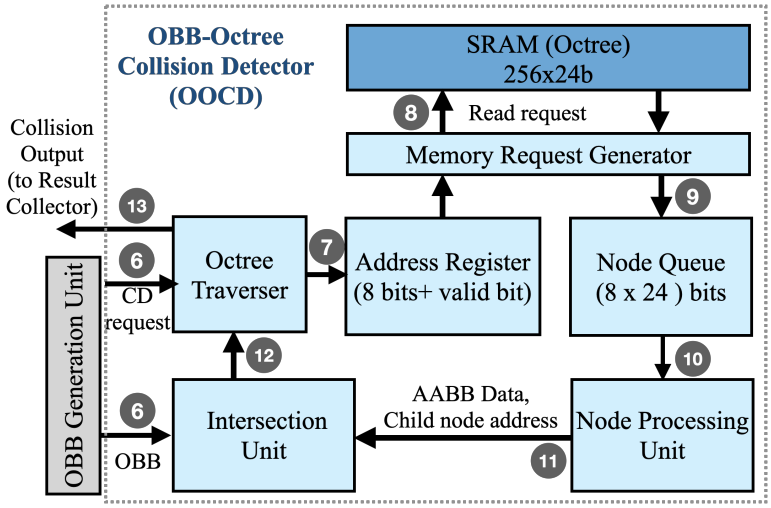


Figure 3.11: Microarchitecture of the CECDU.

from all OOCDs and sends the final collision detection result (True or False) to the scheduler once collision detection for all OBBs of the robot is done. The Result Collector stops collision detection for a given pose if an OOCD returns true for collision detection between OBB-environment. The above architecture performs robot-environment collision detection. Self-collision can be implemented using link-link collision check [242] (using separating axis test for OBB-OBB intersection) or



(a) OBB Generation unit.



(b) OBB-octree Collision Detector.

Figure 3.12: (a) and (b) represents the microarchitectures of OBB Generation Unit unit and OBB-octree Collision Detector (OOCD).

other existing self-collision check approaches [**? ?**].

Figure 3.12a represents the microarchitecture of the OBB Generation Unit. For each link, the size of its bounding box, and the radii of upper and lower bounding spheres are stored in the SRAM. At runtime, the OBB Generation Unit receives the robot's pose ❶. The process of converting a robot's pose from its C-space to the physical space (i.e., location and orientation of robots' body parts) is known as forward kinematics. The transformation matrix (i.e., rotation and translation) for each rigid body of the robot for a given pose can be generated using forward

kinematics [52, 183]. The Transformation Matrix Generator is configured with the parameters needed to calculate a transformation matrix (4×4) for each link for this pose (e.g., length of a link, distance between two joints). This matrix is used to find the rotation and translation of a robot link's bounding box (i.e., OBB). The transformation matrix requires the sine and cosine of the joints' angle values for rotational joints. A trigonometric function unit ❷ is used for sine/cosine calculation for transformation matrix generation. We use a fifth-order approximation-based trigonometric function unit [50]. The trigonometric function unit is a 5-stage pipelined unit consisting of 8 multipliers, 3 adders/subtractors, and registers. The transformation matrix of each link is then sent to a matrix multiplier and adders ❸. These ALUs calculate the center and orientation of the OBB for this link ❹. Thus the OBB Generation unit generates a set of OBBs to represent the space occupied by the robot for its given pose and sends the OBBs to OOCDs for collision detection ❺. Each OBB is represented by 17 values (16-bit each), 3 for its center. 3 for its size, 9 for its 3×3 orientation, and 2 for radii of the bounding and inscribed spheres.

Figure 3.12b represents the microarchitecture of the OOCD. The Octree Traverser (a finite state machine) receives the collision detection request from the OBB Generation Unit ❻ and stores the root node's address (i.e., 0) to the Address Register and sets its valid bit ❼. The SRAM stores the environment octree. The Memory Request Generator sends a memory request when the Address Register has a valid entry ❽. The received data is then added to the Node Queue ❾. The Node Queue can store 8 entries with 24 bits per entry. The Node Processing Unit receives this node information ❿. Here, each node represents an AABB in the space and contains the occupancy information of its octants. An octant can be empty (i.e., no obstacle in this space), partially occupied, or fully occupied. The node information (24 bits) consists of occupancy information of all octants and the addresses for children nodes corresponding to partially occupied octants. The Node Processing Unit uses the node information and sends intersection queries for occupied octants ⓫. Each query consists of the AABB information corresponding to an octant and the address of the child node. An AABB is represented by its center and size (6×16 bits). The Intersection Unit performs an AABB-OBB intersection test using the proposed cascaded early-exit intersection test flow (Figure 3.10). We explore pipelined and multi-cycle designs for the Intersection Units. The Node

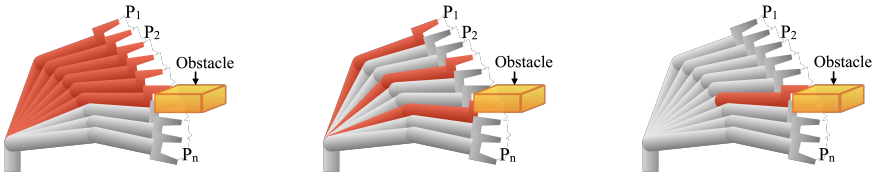
Processing Unit sends one query every cycle for pipelined intersection units. For a multi-cycle unit, it sends a query when the Intersection Unit is free. The Intersection Unit consists of fixed-point multipliers and adders. The Octree Traverser receives the intersection test output (0/1), and the child node address (8 bits) (12). If a collision is found for a partially occupied octant, the address for the corresponding child node is stored in the Address Register (7). The Octree Traverser sends back the collision detection result (True or False) to the Result Collector (13) upon traversal completion.

3.4 Collision Prediction for Motion Planning

SAS focuses on exploiting inter-collision-detection parallelism to accelerate collision detection for motion planning by reducing redundant computation. It takes advantage of the physical spatial locality of different poses of the robot to increase the probability of finding a colliding pose in a given motion with fewer collision detection checks. In this section, we further explore the potential of collision prediction to reduce the number of collision detection queries performed during motion planning. This section describes the key insight for collision prediction and provides a limit study to motivate collision prediction in motion planning. Further, we motivate the use of a hashing function and collision history table and explain the proposed collision prediction approach, COORD , in detail. Finally, we explain the microarchitecture of the proposed collision prediction unit (COPU) in detail.

3.4.1 Collision Prediction Limit Study

As explained in Section 3.1, a motion-environment or pose-environment collision check is carried out by executing multiple smaller CDQs (e.g., OBB-environment) and combining the output of all CDQs using an `OR` operation. Thus, if any of these CDQs returns `True` for collision, the entire motion is determined to be in a collision, and execution of subsequent CDQs is skipped. For a collision-free motion/pose, the order of CDQs execution does not matter, as all CDQs are performed before concluding that the motion/pose is collision-free. However, for colliding motions/poses, the execution order of CDQs affects the computation performed. Our profiling of 1000 motion planning queries covering different motion



(a) Naive scheduling

(b) Coarse-step scheduling

(c) Oracle scheduling

Figure 3.13: (a)-(c) represent different scheduling orders of pose-environment CDQs for a motion-environment collision check.

planning algorithms and robots shows that $52\% - 93\%$ motions checked for collision during motion planning are colliding (methodology and environmental scenarios explained in Section 4.5). Thus, there is a potential to reduce the collision checking time for these colliding motions by focusing on the order of CDQs.

Figure 5.1 compares three different CDQs scheduling/ordering approaches to demonstrate their impact on the number of CDQs executed. Figure 5.1 represents a short motion of a robot with three links. This short motion is discretized into nine poses. Here, each pose is represented using three OBBs (i.e., three CDQs per pose). This short motion can be checked for collision with environment by executing the resultant 27 CDQs. For naive ordering of CDQs, all poses are checked for collision serially (Figure 3.13a), and 18 CDQs are executed before concluding that the motion collides with the obstacle. For the proposed spatially-aware scheduling policy CSP (Section 3.2) with a step size of 3, poses are checked in the order P_1, P_4, P_7,..,P_2,P_5,...P_n. Thus, with CSP-based ordering of robot poses and CDQs, 9 CDQs (for three poses) are executed before finding a collision, as shown in Figure 3.13b. In contrast, an Oracle predictor knows the collision outcome of all CDQs, and it can order a colliding CDQ first and rule out this colliding motion by performing fewer CDQs. Thus, an Oracle predictor needs to execute only one CDQ for a colliding motion/pose, as shown in Figure 3.13c. Thus, a predictor has significant potential to reduce collision detection computation (e.g., number of CDQs executed).

We perform a limit study of the potential reduction in the number of CDQs performed during motion planning using three motion planning algorithm-robot combinations (methodology described in Section 3.6). Figure 3.14 compares the

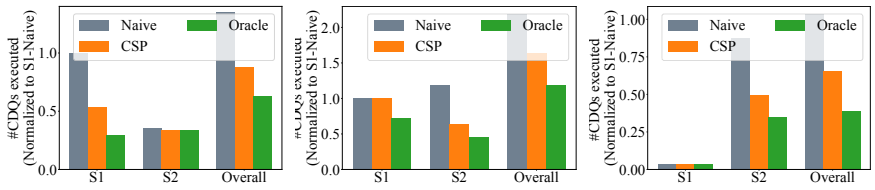


Figure 3.14: (a)MPNet [198]-Baxter robot [208], (b) GNNMP [282]-KUKA robot [3], (c)BIT* [74]-2D environment. compare the effect of different scheduling policies on the number of CDQs executed for motion planning queries.

number of CDQs executed using a naive, CSP-based, and Oracle prediction-based ordering of CDQs. We have divided each motion planning algorithm into two stages, labeled S1 and S2, based on the type of CDQs performed in these stages. For example, in S1 of MPNet, different motions are explored to find a suitable and short path to the end goal. In this exploration stage, the majority of the motions explored are colliding. Whereas in S2, the trajectory (i.e., set of motions) determined by S1 is checked for feasibility. The majority of the motions checked for collision in this stage are collision-free. Collision prediction reduces the computation for a colliding motion by prioritizing checking parts of the trajectory that are more likely to collide with environmental obstacles. However, collision prediction does not reduce computation for a collision-free motion. Thus, collision prediction provides a higher reduction in the number of CDQs performed in S1 (44%) compared to S2 (0.02%), as most short motions checked for collision in S2 are collision-free. Overall, Oracle prediction results in 29.7%, 25.1%, and 40.7% reduction in the number of CDQs compared to CSP ordering. Further, we measure the performance/watt on a collision detection hardware accelerator. Our evaluation using microarchitectural simulators suggests that an Oracle predictor (100% precision and recall with zero cycle latency) can provide $1.11 - 1.44\times$ increase in performance/watt for different motion planning algorithms and 7-DOF robotic arms (methodology described in Section 3.6).

We further analyze the advantage of Oracle prediction for varying difficulty levels in motion planning. The evaluated benchmarks consist of environments and

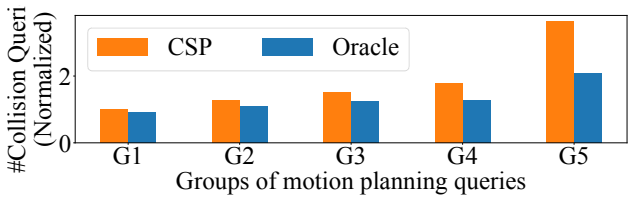
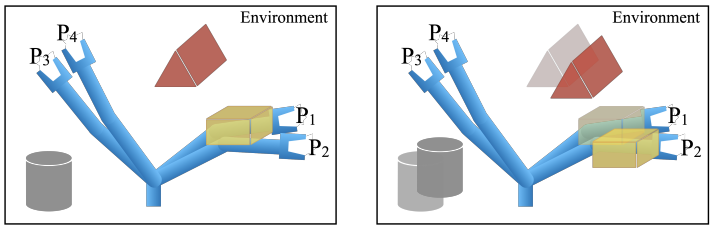


Figure 3.15: Collision prediction for GNN motion planning for a 7-DOF KUKA robot. Here, G1-G5 represent different difficulty levels of motion planning queries, G5 being the highest.

motion planning queries of varying difficulty levels. For cluttered environments (e.g., more obstacles) or difficult motion planning queries (e.g., passing through narrow area), motion planning typically requires more time as it needs to check multiple possible motions and poses to find a collision-free trajectory. We use the number of CDQs performed during a motion planning query to approximate its difficulty level and divide the benchmarks into five equal-size groups, G1-G5, where the difficulty level increases from G1 to G5. Figure 3.15 compares the number of CDQs for CSP and Oracle prediction. For groups G1-G5, Oracle predictor achieves 9%, 14.3%, 18.7%, 28.3%, and 42.5% reduction in the number of CDQs compared to CSP, respectively. This shows a higher potential for improving motion planning for cluttered and challenging environments. Service and assistive robots suitable for healthcare and older adults must perform motion planning in cluttered environments in real time [134, 166]. Several works focus on improving motion planning in cluttered and highly dynamic environments as it is a challenging problem [111, 124, 151, 227]. We observe that the potential reduction in computation and runtime collision prediction increases as the difficulty level of a motion planning query increases.

3.4.2 Physical Spatial Locality for Collision Prediction

The motion planning pipeline is repeatedly executed for a robot operating in dynamic environment. During a single execution obstacle occupancy is typically considered fixed. During this single execution several collision detection checks are performed for different possible poses and motions. An important consequence of this process that we exploit is that several CDQs are performed for different robot poses with



(a) Intra-frame Spatial locality

(b) Inter-frame Spatial locality

Figure 3.16: (a) compares collision outputs of four different poses of a robot in a given environmental scenario. A robot's physically nearby poses tend to have the same collision output due to physical spatial locality. (b) shows the impact of the temporal-spatial locality of obstacles on collision output for nearby poses to demonstrate the use of collision history from previous time frames.

the same occupancy information. Thus, we can use the collision outcome history of executed CDQs during a query to predict outcomes for next CDQs.

We observe that the physical spatial locality of the robot's poses can be used for collision prediction. Figure 3.16a represents four different poses of a robot in an environmental scenario with three obstacles. Here, pose P_1 is in collision with an obstacle. It is more likely that pose P_2 physically close to P_1 is also in collision with this obstacle. Similarly, if P_3 is collision-free, it is more likely that its nearby pose P_4 is also collision-free. Figure 3.16b represents the environmental scenario for the next time frame and motion planning query. Depending upon the speed of obstacles and the time between two motion planning queries, some temporal-spatial locality exists in the space occupied by obstacles. So if pose P_1 was in collision in the previous frame, it is more likely that a physically nearby pose P_2 is also in collision in the current frame.

These examples demonstrate that the history of collision detection outcomes of a robot's poses can be used to predict the collision detection output of other physically nearby poses.

3.4.3 Collision Prediction in Robot's Configuration Space

We first explore the possibility of applying a hashing function directly to the C-space representation of a robot's pose. As mentioned in Section **??**, robot motion planning is performed in the C-space of a robot, and C-space representations of different robot poses are converted to physical space representations (e.g., a set of geometric primitives covering the space occupied by a pose or motion) using robot kinematics. Collision prediction using the C-space representation of a robot pose does not require C-space to physical space representation transformation before collision prediction, thus saving computation.

The C-space representation of an n-DOF robot's pose is an n-dimensional real-valued vector, where each value represents a DOF. Each DOF of a robot is represented using a continuous value (e.g., angle of its joint), and thus, the space of these poses is large. As explained in the previous Section, we can use the physical spatial locality of different poses for collision prediction. Based on this insight, a collision history table can be used for collision prediction by grouping physically nearby robot poses using a hashing function. This strategy results in a small and dense collision history table. Next, we discuss different hashing strategies for C-space representation of a robot's pose. These hashing functions are designed with the aim of grouping physically nearby positions of a robot.

POSE: For POSE, the hashing function is applied to the robot's pose in the C-space. This pose is an n-dimensional vector for an n-DOF robot, where each value represents a DOF. Each value in this pose is quantized and converted to k bits, giving a hash code of kn bit-width.

POSE-part: Next, we consider a hash function POSE-part that is only applied to x DOFs closest to the base of the robotic arm. Figure 3.17b represents two poses, P_1 and P_2, where the first joint (closest to the base) is changed by $\pi/8$. This small change in the first DOF results in a significant change in the space occupied by both poses. In contrast, Figure 3.17a represents two poses where the last DOF (farthest from the base) is changed by π between two poses. However, there is a significant overlap between the spaces occupied by poses P_1 and P_2. These examples demonstrate that for robotic arms with multiple links, the DOFs closer to the base have a higher impact on the physical closeness between two poses. In the POSE-part

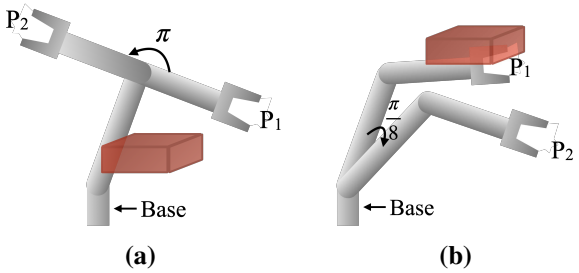


Figure 3.17: (a) and (b) compares the effect of change in a DOF on the space occupied by the robot. A DOF closer to the robotic arm's base has a higher impact on the space occupied by the robot, as shown in example (b).

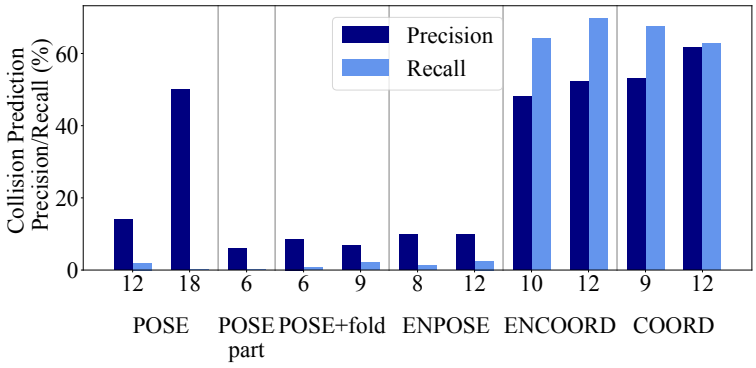
hashing function, we only consider the first two DOFs, which significantly reduces the size of the hash table.

POSE+fold: POSE results in long hash codes (kn), which can result in large and sparse collision history. We further explore the use of folding on the POSE hash code to reduce the size and sparsity of the collision history table. In POSE+fold, a part of the POSE hash code is XORed with the other part.

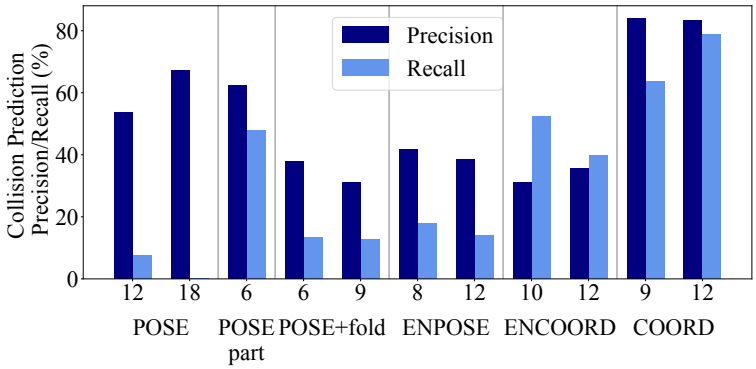
ENPOSE: One approach to reducing the size of the hash code is to use a fixed-size latent space representation of the robot's pose. We train a simple encoder-decoder on $32,768$ random poses using the loss between input poses and decoded poses. The encoder and decoder are simple one-layer MLPs to keep encoding computation overhead low. We explore 2 and 4-dimensional latent space representation and quantize latent space representation to generate hash code.

Figure 3.18 compares the precision and recall of different hashing functions for low and high-clutter environments (Methodology described in Section 3.6). The collision prediction strategy is described in Section 3.4.5. Here, the collision prediction precision represents the fraction of poses in collision from poses predicted for collision. Collision prediction recall is the ratio of the number of colliding poses predicted to be in a collision by the predictor and total colliding poses.

We observe that even though the precision of POSE is high, its recall is very low due to large and sparse collision history table. For example, the collision history table size is 2^{18} for a hash code with 18 bits. For each benchmark, 1000 CDQs



(a) Low obstacle density environments



(b) High obstacle density environments

Figure 3.18: (a) and (b) compares the collision prediction precision and recall of different hash functions for environments with low and high density of obstacles, respectively. Random baseline precision is 2.6% for low-density and 26% for high-density environments. POSE: Hash function applied to the pose of the robot in configuration space, POSE+fold: Hash function applied to the pose with hash folding using XOR, ENPOSE: Hash function applied to the encoded pose, COORD: Hash function applied to Cartesian coordinates of the centers of individual links of the robot.

are performed, which sparsely fills the collision history table. The recall slightly increases for the hash code size of 12 bits. However, the precision degrades due to coarse-grained binning. POSE+fold reduces the hash code size and increases the recall at the cost of precision. This is due to the folding process not preserving

physical spatial similarity between two poses. In contrast, POSE+part increases the precision and recall as it preserves the physical spatial locality for links closer to the robot's base. ENPOSE results in very low precision (close to baseline using random prediction). We believe that it is because latent space representation does not preserve physical spatial locality. *In summary, C-space representation hashing does not provide sufficient precision and recall as simple hashing strategies do not capture physical spatial locality in hash codes.*

3.4.4 Collision Prediction in Physical Space

In this section, we explore the potential of using physical space representations of a robot and environmental obstacles for collision prediction. Each CDQ is performed between a part of the robot and the environment. We propose to apply a hashing function to the space represented by a robot. For example, for a given robot pose, the hashing function can be applied to the physical location of the center of each link. We also discuss how collision prediction can be used by applying the hashing function to the space occupied by environmental obstacles in Section **??**.

The C-space representation of the robot's pose is used to find a transformation matrix for each rigid link of the robot [225, 242]. This transformation matrix can be used to calculate geometries for each link (e.g., OBBs, spheres). This transformation matrix is a 4×4 matrix, which represents the rotation and translation of each link for a given pose [52]. Thus, this transformation process provides the Cartesian coordinates of the center of different rigid parts of the robot, which can be used to generate hash code. For example, a 2-DOF robot in Figure 3.17 consists of 3 rigid parts. In the proposed approach **COORD**, the hashing function is applied to the Cartesian coordinates of the center of each link. These hash codes are used to make collision predictions for each link, which can be used to prioritize execution CDQs corresponding to a link if a collision is predicted. We consider OBB-based [11, 225] and spheres-based [242] representation of a link in evaluation. The center of a link is represented using three 16-bit fixed point representations of its Cartesian coordinates. Figure 3.19 represents hash code generation from a link's center coordinates. The most significant bits (depending upon the size of the hash code) from each coordinate are selected for hash code generation. Thus, this hash function

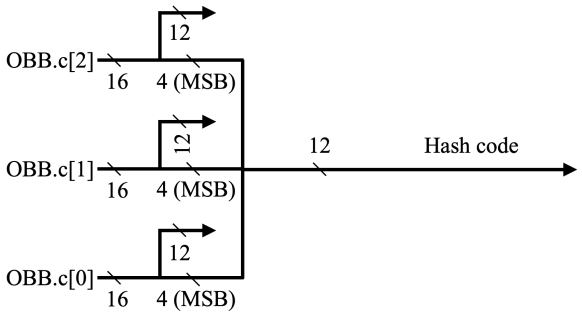


Figure 3.19: Hash code generation for a robot link using COORD hash function on its center link.c.

preserves the physical spatial locality of links' centers for hash code generation.

We further explore the application of hashing function to the latent-space representation of the Cartesian coordinates of the center of a link. We use simple one-layer MLP for encoding, and use 2 and 4-dimensional latent space representation and quantize latent space representation to generate hash code. This approach is referred to as **ENCOORD**.

Figure 3.18 compares the precision and recall of the ENCOORD and COORD hash function. ENCOORD achieves comparable precision and recall for low-clutter environments. However, the precision and recall reduce for high-clutter environments. We believe that latent-space representation does not completely preserve the physical spatial locality. For high-clutter environments, this result in frequent inaccurate updates of collision history, which leads to lower precision and recall. The figure shows that COORD results in the highest collision prediction precision and recall. COORD provides 77% precision with 47% recall even for low-clutter environments. *Overall, COORD provides good precision and recall across low- and high clutter environments, and is used as the hashing function for collision prediction in this dissertation.*

3.4.5 Collision History Table Update and Prediction Strategy

The Collision History Table is updated with the collision outputs as CDQs are performed. Each entry in the Collision History Table maintains saturating counters for colliding (`COLL`) and collision-free (`NONCOLL`) queries observed in the past.

We observe that updating the history table for CDQs for all colliding queries is important for prediction precision and recall. However, we can reduce the frequency of updates with collision-free CDQs. We use a parameter U to define the update frequency for collision-free CDQs. For every N collision-free CDQs executed, $N \times U$ CDQs are chosen randomly to update the Collision History Table. A lower value results in lower traffic and updates of the Collision History Table.

The collision prediction strategy determines when a collision is predicted for a hash code given the Collision History Table entry with counts `COLL` and `NONCOLL`. The proposed collision prediction strategy predicts a query to be colliding if `COLL` $> S \times$ `NONCOLL`. Thus, the value of parameter S sets the weight of `NONCOLL` for prediction and determines the aggressiveness of the collision predictor. A lower value of S means a more aggressive predictor. Note that for $S = 0$, the Collision History Table does not need to be updated for collision-free CDQs (i.e., $U = 0$). Further, each entry requires only one bit for this case. Section 3.7.3-3.7.3 compare the effect of different values of S and U on precision and recall.

3.4.6 Collision Prediction and Detection on CPU and GPU

Our goal is to use collision prediction to reduce the computation required for motion planning collision detection, which can be exploited to reduce runtime and improve energy efficiency by reducing dynamic power consumption. We first evaluate the impact of collision prediction for collision detection using CPU and GPU (methodology given in Section **??**). The motion planning benchmark consists of several motions to be checked for collision. Algorithm 1 represents a pseudo code for motion-environment collision detection with collision prediction. Four threads are used for CPU-based implementation, where each thread executes Algorithm 1 for a group of motion. For GPU-based implementation ($>= 512$ threads), one block is allocated per motion (CUDA programming model), and poses within a motion are allocated to different threads. In this case, each thread executes Algorithm 1 for a subset of poses in a motion. The hash table is shared between all threads.

For CPU-based implementation, we observe 25.3% reduction in the number of collision detection queries (i.e., computation) and 13.8% reduction in the collision detection runtime. Our profiling suggests that the collision history table accesses and

updates increase cache misses, which might explain the gap between computation and runtime reduction. Collision prediction on CPU accelerates collision detection by 13.8%. However, this speedup might not be sufficient to enable real-time motion planning execution on a CPU.

Algorithm 1 Motion collision detection with collision prediction

Input: Motion_info = [pose1, pose2,...,poseN], Hash_table, Parameter S ;
Output: Collision;

```
1:  Queue=[]; Collision=False
2:  for pose ∈ Motion_info do
3:      OBBs = robot_kinematics(pose)
4:      for OBB ∈ OBBs do
5:          hashentry=hashcode(OBB.c)
6:          if hashentry.COLL > (hashentry.NONCOLL >> S) then
7:              CollTemp = Collision_check(OBB)
8:              Collision = Collision OR CollTemp
9:              Hash_table.update(hashcode(OBB.c),CollTemp)
10:             if Collision then return True
11:             end if
12:         else
13:             Queue.append(OBB)
14:         end if
15:     end for
16: end for
17: if Collision==False then
18:     for OBB ∈ Queue do
19:         CollTemp = Collision_check(OBB)
20:         Collision = Collision OR CollTemp
21:         Hash_table.update(hashcode(OBB.c),CollTemp)
22:         if Collision then return True
23:         end if
24:     end for
25: end if
```

We further evaluate the impact of collision prediction on the number of CDQs executed and runtime for different levels of parallelism in GPU. Figure 3.20 represents the number of CDQs executed for MPNet motion planning with and without collision prediction. Number of CDQs are normalized with respect to CDQs executed for 64 threads configuration with collision prediction. It also represents normalized runtime (with respect to runtime for 4096 threads) for collision detection with prediction. As the degree of parallelization increases, baseline collision detection results in higher CDQs as redundant work increases due to the parallel execution of CDQs within a motion. However, collision prediction helps with reducing redundant work by prioritizing the execution of CDQs that are likely to result in positive collision output. Further, we find that collision prediction-based

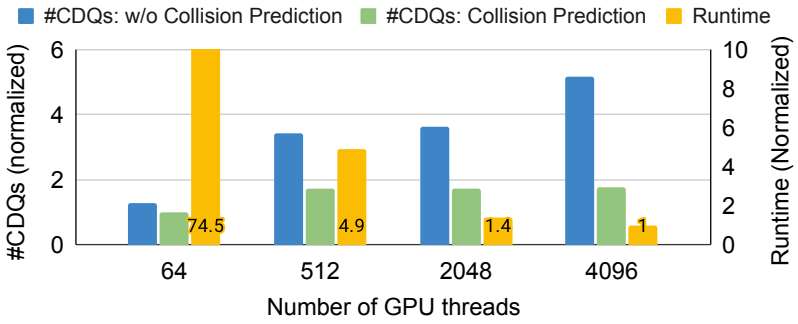


Figure 3.20: Impact of collision prediction on numbers of CDQs executed for GPU-based collision detection.

execution results in 30% (2048 threads) to 70% (4096 threads) increase in the execution time. Our profiling results suggest that software collision prediction increases warp divergence due to skipped computation and memory stalls due to hash table accesses.

The use of application-specific accelerators is desirable to meet the runtime and energy constraints of robot motion planning, especially for high-DOF robots working in dynamic environments [11, 95, 172, 225]. We further explore collision prediction integration with collision detection hardware accelerators. We find that software-collision prediction (CPU controller) is $\sim 2\times$ slower than collision detection using a specialized accelerator [11, 225], which results in an overall $\sim 2.1\times$ collision detection slowdown. This motivates the need for a collision prediction accelerator that can be integrated with existing collision detection hardware accelerator design. In the next section, we propose a hardware collision prediction unit and discuss its microarchitecture.

3.4.7 Collision Prediction Unit Microarchitecture (COPU)

This section describes the microarchitecture of the proposed Collision Prediction Unit (COPU) and its integration with Collision Detection Units (CDU) for motion planning. Figure 3.21 represents detailed architecture of collision prediction's integration with CDUs. The gray colored blocks represent baseline architecture and yellow-colored blocks represent additions for collision prediction. Here, we use the

proposed CECDU (Section 3.3) as baseline hardware collision detection system. The collision prediction unit and OBB-generation unit is shared by multiple CDUs. Each CDU performs intersection test between environmental obstacles and a part of robot (e.g., OBB or Spheres). CDUs represented in Figure 3.21 corresponds to the OBB-Octree collision detection units (OOCDs) in Figure 3.11. Note that the proposed COPU can be integrated with any collision detection accelerator. Further, the proposed collision predictor augmented collision detection unit can be part of a motion planning acceleration systeml, such as MPAccel explained in Section 3.5.

The OBB Generation Unit receives a pose from a pose-environment collision detection query scheduler, and generates OBBs for each link ❶. The output OBB's center (proxy for link's center) is used to generate the hash code ❷ and read the corresponding entry from the **Collision History Table (CHT)** ❸. The Collision Prediction Unit sends the OBB and prediction to corresponding queues ❹. There are two separate queues `QCOLL` and `QNONCOLL` for storing OBBs with collision predicted and not predicted, respectively. Priority must be given to OBBs stored in the `QCOLL` queue as these OBBs are more likely to return `True` for collision check. The Query Dispatcher sends a query from `QCOLL` queue to a free CDU if it is not empty ❺. However, if `QCOLL` is empty, the Query Dispatcher sends a query from `QNONCOLL` queue only if `QNONCOLL` is full or the OBB generation unit has received all poses from the scheduler ❻. In the latter two cases, no new OBB can be added to `QNONCOLL` queues, and we need to perform collision detection from the `QNONCOLL` queue to create space in the queue. Thus, the Query Dispatcher prioritizes the `QCOLL` queue for collision detection and dispatches from `QNONCOLL` only if necessary. CDUs send the collision output to the Query Update Unit to update the hash table ❼. The Result Collector receives the output of all CDUs and sends the output to the scheduler ❽.

The Collision History Table (CHT) is implemented using an SRAM. Each entry of CHT consists of 4 bits for `COLL` and `NONCOLL` saturating counters. The Query Update Unit reads an entry from CHT and updates `COLL` or `NONCOLL` (based on collision result) using a saturating adder. The Collision Predictor uses OBB's center to generate CHT address using COORD and reads corresponding entry {`COLL`,`NONCOLL`}. The predictor uses comparison (`COLL`> (`NONCOLL`>>x)) to determine collision output, where x is determined using parameter *S* used in

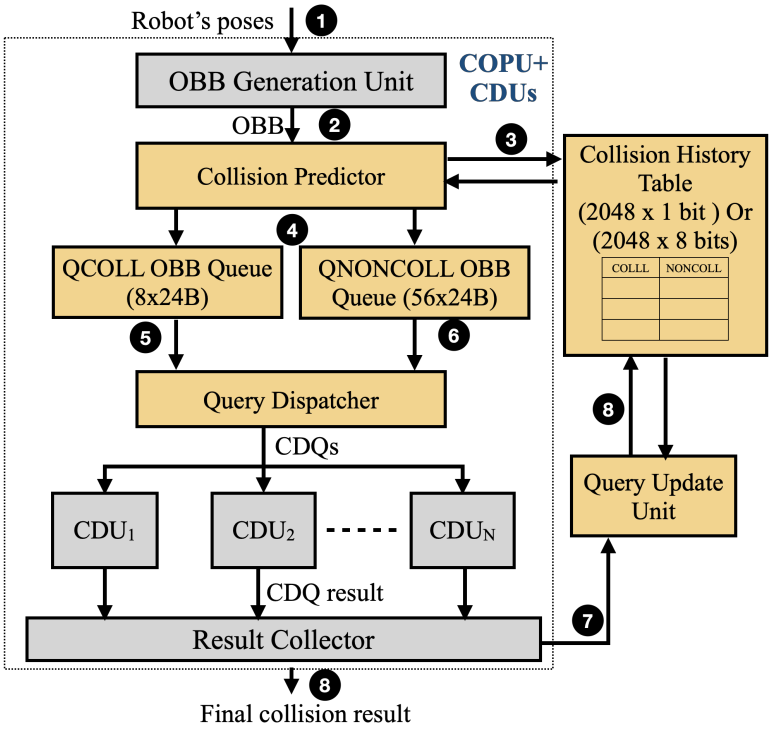


Figure 3.21: Collision detection accelerator architecture and integration of Collision Prediction Unit with CDUs. Here, gray colored blocks represent baseline architecture (CECDU exaplained in Section **??**), and yellow colored blocks represent additions for collision prediction.

collision prediction strategy (Section 3.4.5).

3.5 Motion Planning Hardware Accelerator Architecture

This section describes the microarchitecture of MPAccel, a motion planning hardware accelerator. MPAccel consists of the proposed blocks SAS and CECDU. Figure 3.22 represents the architecture of the overall system for a learning-based motion planning algorithm (e.g., MPNet). The controller receives the environment's occupancy information from sensors (processed by a perception module) and sends it to the DNN accelerator and SAS ❶. The controller receives a motion planning query consisting of the start and end goals. The controller runs the motion planning

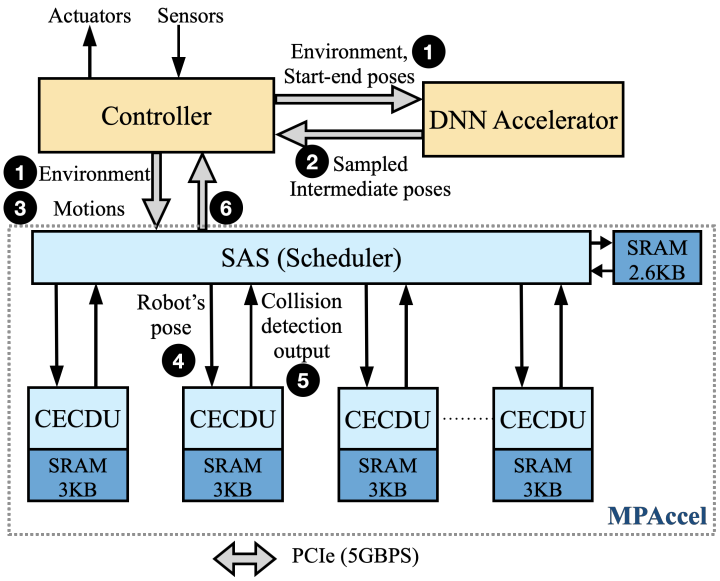


Figure 3.22: Architecture of MPAccel.

algorithm and offloads neural network inferences to the DNN accelerator and collision detection to MPAccel. A simple CPU core can be used as the controller. The DNN accelerator generates intermediate poses for a candidate trajectory between the start and the end goal ❷. The controller receives these intermediate poses and generates a set of motions for collision detection based on the motion planning algorithm ❸. The bandwidth of the bus between the CPU controller and DNN accelerator and SAS is assumed to be 5GBPS, which can be achieved by PCIe [175]. SAS receives the group of motions and function mode from the controller and schedules collision detection queries to the CECDUs ❹. The scheduler also collects results from the CECDUs ❺ and sends back the aggregated result to the controller once the execution finishes ❻.

CECDU calculates the robot's occupied space for different poses on-chip and uses octree representation for the environment, which reduces the storage requirement. We find that on-chip memory of 50KB is sufficient to solve motion planning for high-DOF robots (~ 7). Hence, we use on-chip SRAM for storage, and MPAccel is not connected to DRAM. Prior motion planning accelerators deployed in real-world applications have proposed to use only on-chip memory to meet the energy

and real-time constraints [172, 173, 231, 233].

3.6 Methodology

This section provides a detailed methodology for the evaluation presented in this chapter. We have provided the methodology used for detailed evaluation and analysis of SAS, CECDU, and COPU. Further, we provide the methodology used to evaluate MPAccel for motion planning execution.

We build detailed microarchitectural simulators of SAS, CECDU, COPU, and MPAccel for evaluation. We use Verilog to build RTL models for the SAS, CECDU, and COPU blocks. RTL implementations are synthesized using the Synopsys Design Compiler and the OpenRAM Memory Compiler [91] to estimate the area and power at 45nm technology using FreePDK design library [238]. The timing model of the microarchitectural simulator is based on the cycle latency measured from RTL models. We use 16-bit fixed-point number representation for poses, OBBs, and AABBs.

We evaluate different motion planning algorithms for robots using simulators. We use Klampt [100] and PyBullet [46] robotic simulator to simulate robot's motions ans poses. We further provide details of the benchmarks used for evaluation.

SAS: We use Baxter (7-DOF) robot and collision detection queries generated from MPNet [198] motion planning algorithm for our evaluation. We use ten environmental scenarios with 100 pairs of start and end goals per each environmental scenario. In each sample environmental scenario, the robotic arm is in front of the table, and $5-9$ obstcales of different shapes (e.g., bottles, books, mugs, and boxes) are randomly placed on the table. The size of these obstacles in each dimension is limited to $3\%-12\%$ of the environment's extent. These benchmarks are consistent with other work on motion planning and collision detection [153, 171, 200]. SAS is connected to collision detection units, and CECDUs are used in this setup to account for the collision detection latency in the simulation.

CECDU: We use Kinova Jaco2 [132] (6-DOF) with 7 links for evaluation of CECDU. We use ten environmental scenarios with 100 random poses of the robot.

Each sample environment contains $5-9$ randomly placed cuboid-shaped obstacles. The size of these obstacles in each dimension is limited to $3\%-12\%$ of the environment's extent. For OOCD, Our proposed method reduces energy by exiting early from the intersection test flow (Figure 3.10). Thus the proposed method reduces the switching activity. We built an accurate architectural power model to speed up power measurement of OOCD using the methodology described in [27]. We use RTL simulation to find out the leakage and dynamic power of individual blocks (multiplier, adder, mux) and use the microarchitectural simulator to estimate their activity factors.

COPU: We study collision prediction precision and recall for different hashing functions and design aspects using random environmental scenarios and poses of the robot (Section 3.4.3, Section 3.4.4, and Section 3.7.3). We use a 7-DOF robotic arm Kinova Jaco2 [132] for these benchmarks. We generate an environmental scenario for each benchmark with random placement of $5-9$ cuboid-shaped obstacle. The size of the environment is limited to the reach of the Jaco2 robot. These benchmarks are consistent with previous works on motion planning [153, 172]. 1000 random robot poses are sampled in an environment for collision prediction evaluation. For low, medium, and high obstacle density benchmarks, the size and number of obstacles are limited such that, on average $\sim 2.5\%$, $\sim 10\%$, and $\sim 25\%$ robot poses are in collision.

We explore three motion planning algorithms to evaluate collision prediction (Section 3.7.3). MPNet [198] is a leaning-based motion planning algorithm and uses a neural network for trajectory sampling. We evaluate this algorithm for a 7-DOF robotic arm Baxter and 2D path planning. GNNMP [282] uses a graph neural network for sampling the C-space and path smoothing. Batch Informed Trees (BIT*) [74] is an informed-sampling-based motion planning algorithm. Above two algorithms are evaluated for a 7-DOF robotic arm KUKA [3] and 2D path planning. We use the environment scenarios and motion planning queries used in the prior works on these motion planning algorithms [198, 282]. Each environment scenario typically consists of a work table with several objects randomly placed on the table and in the surroundings. We use a CHT of 2048 entries with 1 or 8 bits per entry for two collision prediction strategies.

We also evaluate collision prediction for CPU (Cortex A57 4-core) and GPU (NVIDIA Titan V) based collision detection system. We use MPNet-Baxter benchmark for the same. We use C++ (with OpenMP) and CUDA programming language for implementation, and Valgrind [174] and nvprof [180] for profiling.

MPAccel: We use MPNet motion planning algorithm [200] to evaluate motion planning runtime for a 7-DOF robotic manipulator Baxter [208] using MPAccel. We use ten environmental scenarios (same as SAS evaluation) with 100 pairs of start and end goals per each environmental scenario. Note that MPNet is used as an example of a state-of-the-art sampling-based motion planning algorithm. We chose MPNet [200] as it has shown significant improvement in motion planning performance and has code available for evaluation. However, MPAccel can also be used for other sampling-based motion planning algorithms. We also evaluate collision detection and motion planning runtime on GPUs (NVIDIA Titan V and Pascal GPU with 256-CUDA cores) and CPUs (Intel i7-4771 8-core and Cortex A57 4-core) (Section 3.7.5).

3.7 Evaluation

This section provides evaluation of the proposed approaches in this chapter. First, we provide detailed analysis of individual blocks SAS, CECDU, and COPU. Lastly, we provide evaluation of a motion planning algorithm on MPAccel, and compare with a CPU and GPU systems.

3.7.1 Spatially Aware Scheduler

We propose a multi-motion coarse-step-based scheduling policy (MCSP) and corresponding microarchitecture in Section 3.2. A sampling-based motion planning algorithm consists of multiple phases (Section 2.1), where a set of motions is sent to the scheduler for collision detection in each phase. We use MPNet algorithm and report the average runtime and energy for an entire set of motions for different schedulers. The proposed CECDUs are used as collision detection units (CDUs) in this evaluation. The group size is set to 16 for inter-motion parallelism based on empirical results.

Figure 3.23 compares the performance and energy of different scheduling policies. The number of collision detection tests is used as a measure of energy. For given benchmarks, the on-chip memory of OOCD is sufficient for collision tests, and there is no memory access coalescing across collision tests. Thus, energy increases linearly with the number of collision detection tests. This increase may not be linear if off-chip memory is required (e.g., for high-resolution environments such as those used in games or physics simulations). However, the insights of this evaluation still hold for such cases. As parallelism increases, MCSP and MP outperform NP and group "useful" work to be dispatched to the CDUs. For eight CDUs, MCSP results in 7× speedup with 6% increase in the energy compared to 3.7× speedup with 83% increase in energy for NP. Similarly, for 16 CDUs, MCSP results in 11.03× speedup with 22% increase in the energy compared to 6.2× speedup with 113% increase in energy for NP. The energy consumption for MCSP is slightly higher than predicted by the limit study (Section 3.2). The limit study assumes zero-latency scheduling and equal latency for collision detection. However, the CDU introduces a delay in receiving results for CD queries. In this delay, the scheduler might schedule more CD queries for a motion to free CDUs even though dispatched queries might return `True` for collision detection.

SAS can schedule up to one CD query per cycle. If the latency of CDUs is less than the number of CDUs, then increasing the number of CDUs does not help with speedup as the scheduler can not dispatch CD queries fast enough. Hence the speedup saturates as the number of CDUs reaches 32. However, as shown in the limit study with the ideal scheduler and CDU (zero latency) in Section 3.2, increasing the number of CDUs beyond 64 does not increase speedup significantly and also increases energy consumption.

Effect of Inter-motion Parallelism:

Group size represents the number of motions used for inter-motion parallelism. We compare the effect of group size on speedup and energy for MCSP in Figure 3.24 for eight CDUs. Smaller group size does not take advantage of inter-motion parallelism and results in higher runtime and energy. As the group size increases, inter-motion parallelism helps with improving the runtime by reducing redundant computation.

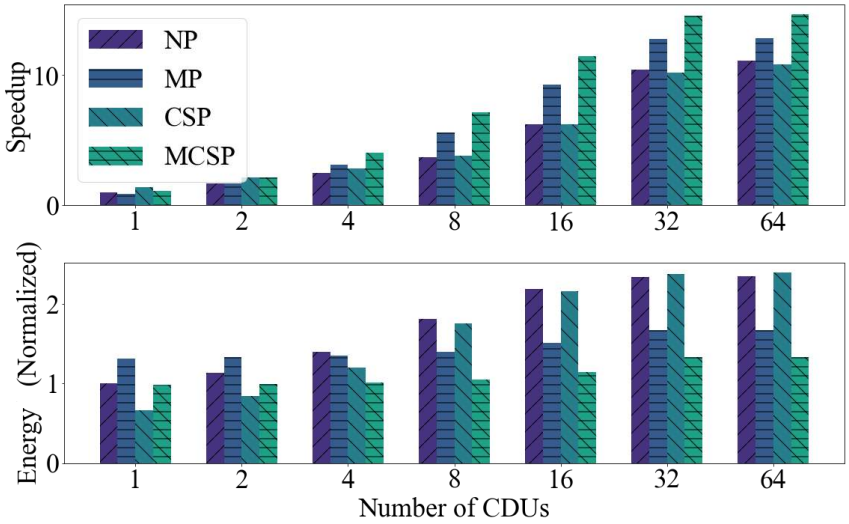


Figure 3.23: Comparison of different schedulers for coarse-grained parallelism. MCSP: Coarse-step policy + inter-motion parallelism (the proposed approach), NP: naive parallelization, CSP: Coarse-step policy, and MP: Only inter-motion parallelism.

The runtime and energy both increase for group sizes greater than 16. Collision detection for a group of motions can be run in different function modes (Section 3.2.1). For example, in the "Connectivity test" mode, once a collision-free motion is found, the subsequent motions can be discarded without checking for collision. More motions are scheduled together as the group size increases, and some motions that could have been discarded are also scheduled for collision detection. This results in the increased energy consumption for larger group sizes.

3.7.2 Cascaded Early-Exit Collision Detection Unit

CECDU consists an OBB Transformation Unit and multiple OOCD for OBB-Octree collision detection test. OOCD also consists of the proposed cascaded early-exit intersection test. First, we evaluate OOCD for OBB-octree collision detection. Further, we evaluate different configurations of CECDU for a 6-DOF robotic arm-environment collision detection.

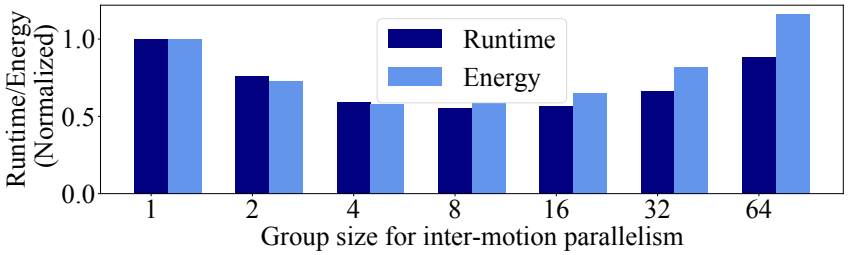


Figure 3.24: Effect of group size for inter-motion parallelism on runtime for MCSP.

Evaluation of OBB-octree collision detection:

First, we compare the latency/computation of the OBB-octree collision detection for parallel and sequential execution of the separating axis test without early-exit approach. Since the intersection test computation is dominated by multiplications, we use the number of multiplications as an estimate of computation. We also provide latency/computation for pipelined and multi-cycle versions. For the baseline version (without checking intersection with bounding and inscribed spheres), parallel execution results in 46% higher computation with $2.52\times$ and $1.77\times$ speedup compared to the sequential execution for multi-cycle and pipelined configurations, respectively. Furthermore, the proposed filter using an inscribed sphere to reduce computation for colliding cases reduces the computation by 33%. The proposed approach based on the bounding and inscribing sphere closes the gap between the computation of sequential and parallel execution, and parallel execution provides $1.2-1.4\times$ speedup with 1.3% more computation. Both filters together provide $\sim 4.1\times$ speedup compared to sequential execution (without sphere-based filters) with 61% computation savings.

Collision detection latency for a robotic arm:

We evaluate the collision detection latency for a 6-DOF (7 links) robotic arm using CECDU. Each CECDU can have more than one OOCD, where each OOCD performs collision detection between an OBB (i.e., robot's link) and the environment. We evaluate two configurations of CECDUs. In the first configuration, the CECDU consists of a single OOCD unit that performs collision detection for all OBBs

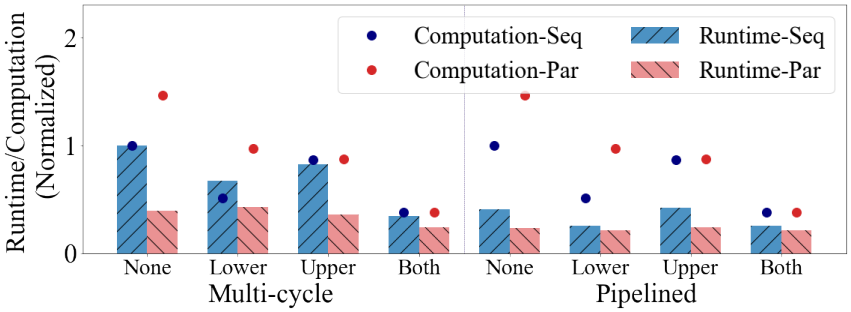


Figure 3.25: Comparison of the runtime and computation for sequential and parallel collision detection.

Table 3.1: Collision detection latency for different CECDU configurations for Jaco2 robot with 7 links and 6 degrees of freedom.

	Single Intersection Unit		Four Intersection Units	
	Multi-cycle	Pipelined	Multi-cycle	Pipelined
Latency (Cycles)	154.4	137.5	54.8	46.3
Area (mm^2)	0.21	0.32	0.69	1.12
Power (mW)	92.6	100.8	215.7	248.7

in a robot serially. If a collision is found between an OBB and the environment, subsequent OBBs are discarded. In the second configuration, the CECDU consists of four OOCD units for parallel collision detection. We also provide evaluation for pipelined and multi-cycle Intersection Units.

Table 3.1 compares the collision detection latency, area, and power for different combinations. Note that the use of four OOCD units does not reduce the runtime proportionally for two reasons. First, subsequent OBBs are not checked for collision in serial execution once a collision is detected. Second, the collision detection time for parallel intersection tests is dominated by the highest intersection test time across all units as we use synchronous scheduling. The end-to-end latency of intersection test is same for pipelined and multi-cycle units. However, the pipelined version can process more than 1 intersection tests at the same time in the pipeline. Therefore, the latency of robot-environment collision detection (which consists of multiple intersection tests) is lower for pipelined version. CECDU performs collision detection for the robotic arm in $46 - 154$ cycles.

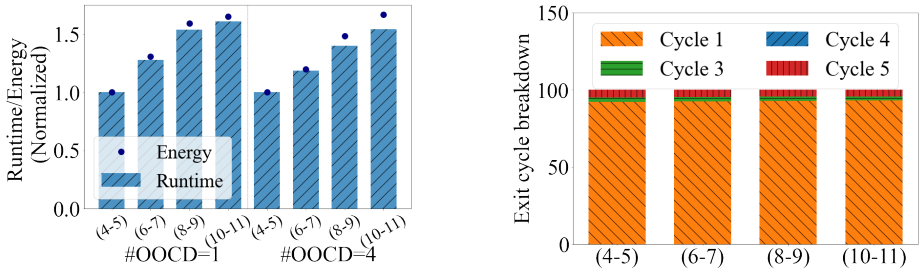


Figure 3.26: (a) represents the runtime and energy for single and four intersection units, and (b) represents the breakdown of the exit cycle from the proposed flow (Figure 3.10) for different environmental complexity (number of obstacles in this example).

We further analyze the effect of environmental complexity (e.g., number of obstacles) on CECDU. Figure 3.26a represents the robot-environment collision detection runtime and energy for environments with increasing number of obstacles. The runtime increases by $\sim 50\%$ as the number of obstacles doubles. Figure 3.26b provides the breakdown of cycles required for the intersection test (Figure 3.10) for different environments. As shown in the figure, the proposed method effectively filters easy cases (cycle-1) across different environmental complexity.

Bakhshalipour et al. [11] also proposed a collision detection acceleration unit CODAcc for OBB-voxelized environment collision detection. In their approach, an OBB is converted to occupied voxels, and read requests for the environment's occupancy information corresponding to these voxels are sent to memory. We could not quantitatively compare our proposed OOCD unit with CODAcc as absolute performance is not reported in their work, and we could not get access to their implementation. Below we provide our insights and approximate comparison of both approaches. Voxelization of OBB results in a simpler intersection test; however, the number of voxels to be checked and memory accesses increase significantly with the resolution of voxelization. Our approximate measurement for the Jaco2 robot shows that for voxels of size 2.56cm (environment's extent is 180cm), the voxelized environment requires 32Kb storage and $30-154$ memory accesses. In contrast, OOCD uses an octree-based compact environment representation and performs

collision detection between OBB-environment in < 40 cycles with 0.75KB on-chip SRAM.

3.7.3 Collision Prediction

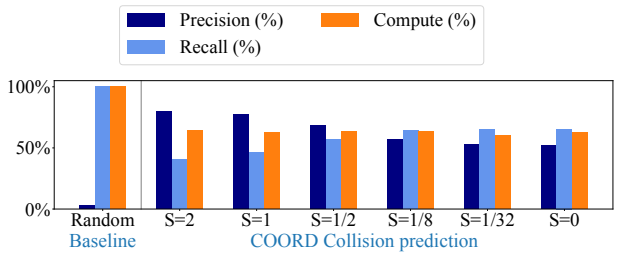
In this section, we present an evaluation of the proposed COORD Collision Predictor and study different design aspects' impact on computation. Next, we evaluate the motion planning runtime, computation, and performance achieved by integrating the proposed COPU with SAS and CECDU.

Collision Prediction Design Aspects

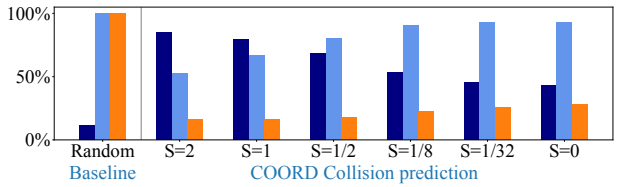
This section provides a detailed evaluation of the proposed COORD Collision Prediction Unit. We explore the impact of different environmental and design aspects on the precision, recall, and potential computation reduction achieved by the collision predictor.

Collision Prediction Strategies: First, we compare different collision prediction strategies for three levels of clutterness in the environment. As described in Section 3.4.5, the proposed predictor predicts collision if $\texttt{COLL} > S \times \texttt{NONCOLL}$, where COLL and NONCOLL represent the counts of colliding and collision-free CDQs for the same hash code (i.e., same entry in the history table). The collision predictor becomes more aggressive with lower values of S. The value of S determines the aggressiveness of the collision predictor. $S = 0$ results in the most aggressive prediction strategy. Figure 3.27 compares the precision and recall for different values of S across different environments. We also report approximate computation reductions achieved by collision prediction using a statistical model. This statistical model considers the baseline collision probability, prediction precision, and recall and provides the potential decrease in the number of CDQs executed for collision check of a motion consisting of 80 discrete poses, and each pose requires 1 CDQ. As shown in Figure 3.27, as the value of S decreases, the predictor becomes more aggressive, resulting in higher recall and lower precision. Precision and recall are important for effectively reducing the computation using collision prediction.

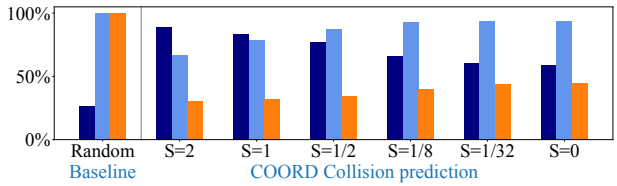
We observe interesting trends in computation reduction across different types



(a) Low obstacle density environments



(b) Medium obstacle density environments



(c) High obstacle density environments

Figure 3.27: (a)-(c) compares the collision prediction precision, recall, and resultant decrease in the computation compared to a random baseline for different collision prediction strategies. The proposed predictor predicts collision if $\texttt{COLL} > \text{S} \times \texttt{NONCOLL}$, where COLL and NONCOLL represent the counts of colliding and collision-free queries in a hash table entry.

of environments. For low-clutter environments, the number of colliding poses is very low, and an aggressive predictor with higher recall is needed to find these colliding poses using prediction. Hence, for low-clutter environments (Figure 3.27a), the recall is more important than precision, as the computation reduction is the highest for $S = 0$. Whereas, for highly cluttered environments, several poses are in a collision. Hence, even though recall is low, the probability that the predictor will find a colliding pose in the motion is high. Here, precision is more important to reduce

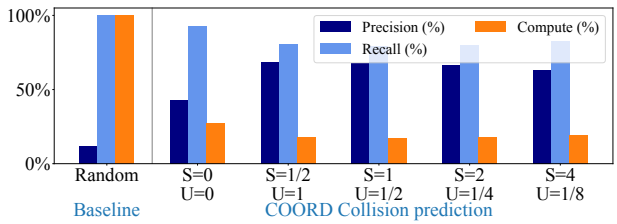


Figure 3.28: Effect of the update frequency of the Collision History Table on collision prediction.

redundant computation. Hence, we observe that for highly cluttered environments, the computation is the least for the highest precision, not recall ($S = 2$). For medium-clutter environments, a balance between precision and recall results in the least computation ($S = 1/2$). We observe that the computation reduction is less sensitive to S compared to precision and recall. In this work, we use a constant value of $S = 0.5$ across all environmental scenarios. However, there is scope to tune the value of S by using a heuristic to estimate environmental obstacle density (e.g., the number of voxels, number of nodes in octree); we leave this to future work.

Collision History Table Updates: For collision prediction with $S > 0$, the hash table needs to be updated with the outcomes of all CDQs executed. More than 90% of the CDQs are collision-free for low or medium-cluttered environments. Here, we explore the effect of reduced update frequency for collision-free CDQs. Figure 3.28 compares precision, recall, and computation for different combinations of S and update frequency U. Here, for all combinations, the computation decrease does not vary significantly ($\pm 1\%$), which suggests that in such cases, the value of S can be adjusted to reduce update frequency.

Collision History Table for Temporal Locality: As mentioned in Section 3.4.2, there is significant temporal-spatial locality between consecutive motion planning queries. Figure 3.29 compares the effect of keeping values in the history table from the previous frames for two different scenarios. Here a scaling factor is used to update the `COLL` and `NONCOLL` counts stored in the history table between two time-frames. A scaling factor of 0 means that all values in the history table are reset

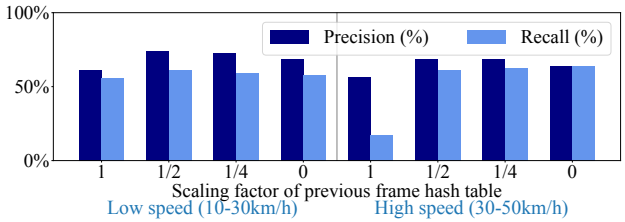


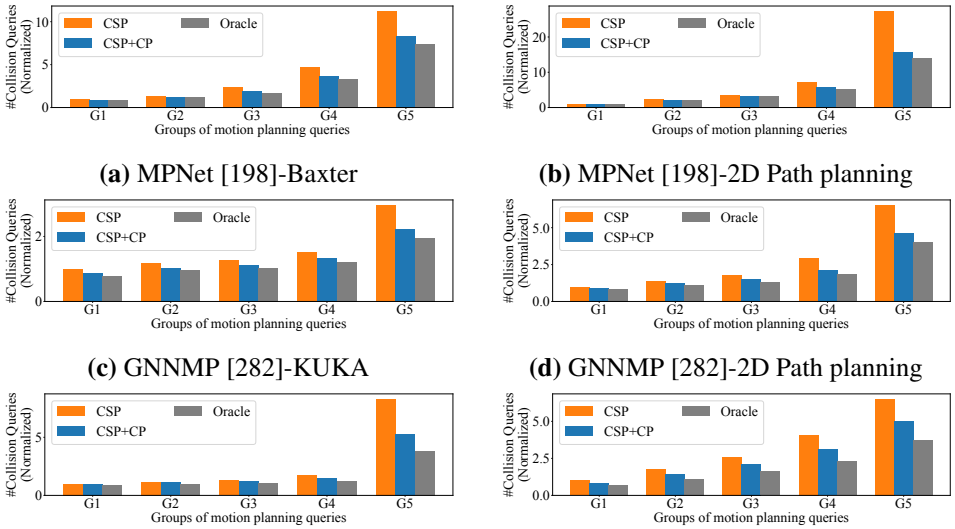
Figure 3.29: Effect of temporal locality and the previous frame's collision history's usage on accuracy and coverage.

to 0 between two frames. Similarly, a scaling factor of 1 means that the current frame's entries are added to the previous frame. We explore scaling factors of $1/2$ and $1/4$ as this can be achieved by shifting the values stored in the history table. As shown in the Figure, the scaling factor of $1/2$ slightly improves the accuracy and coverage in both scenarios. It is more beneficial for scenarios with slowly moving obstacles. In both cases, a scaling factor of 1 results in the worst accuracy and coverage, as it results in slower learning of recent collision trends.

Collision Prediction for Motion Planning

We integrate the proposed collision prediction unit with a collision detection unit as described in Section 3.4.7 for accelerating motion planning execution. We evaluate the reduction in the number of CDQs performed during motion planning for this setup. Figure 3.30 compares the number of CDQs executed for MPNet-Baxter, GNNMP-KUKA, BIT*-KUKA, MPNet-2D planning, GNNMP-2D planning, and BIT*-2D planning. For each motion planning algorithm-robot combination, benchmarks are grouped (G1-G5) according to the difficulty level. We use the number of CDQs performed during a motion planning query to approximate its difficulty level and divide the benchmarks into five equal-size groups, G1-G5, where the difficulty level increases from G1 to G5. Here, all numbers in a plot are normalized to #CDQs for G1 benchmarks and CSP scheduling.

COORD-based Collision Prediction Unit provides 22.5%, 17.4%, 25.5%, 32.06%, 23.1%, and 21.6% reduction in the number of CDQs executed on average compared to CSP-based ordering across different benchmarks (Figure 3.30a-3.30f). For a more difficult problem in a cluttered scenario, the motion planner needs to perform



(a) MPNet [198]-Baxter

(b) MPNet [198]-2D Path planning

(c) GNNMP [282]-KUKA

(d) GNNMP [282]-2D Path planning

(e) BIT* [74]-KUKA

(f) BIT* [74]-2D Path planning

Figure 3.30: Collision prediction for motion planning for different robotic arms.

more tests to find a collision-free motion, which results in higher computation for such challenging scenarios. We observe that the proposed collision prediction unit provides higher benefits for such challenging scenarios. COORD-based Collision Prediction Unit provides 25.9%, 25.4%, 37.2%, 42.9%, 29.3%, and 23.4% reduction in the number of CDQs executed for group G5 across different benchmarks (Figure 3.30a-3.30f).

Performance Evaluation: We analyze the impact of collision prediction on the overall runtime and energy consumption of all collision detection queries for MPNet motion planning for the Baxter robotic arm. We consider two configurations of the collision detection accelerator shown in Figure 3.21. In Config-1, one OBB-environment CDU (i.e., OOCD) is used with one COPU. In Config-2, seven OBB-environment CDUs are used with one COPU. For Config-1, COORD collision prediction results in 23.4% reduction in the energy consumption of collision detection with 1.31× speedup (motion collision detection time) compared to baseline. For Config-2, COORD collision prediction results in 23.4% reduction in the

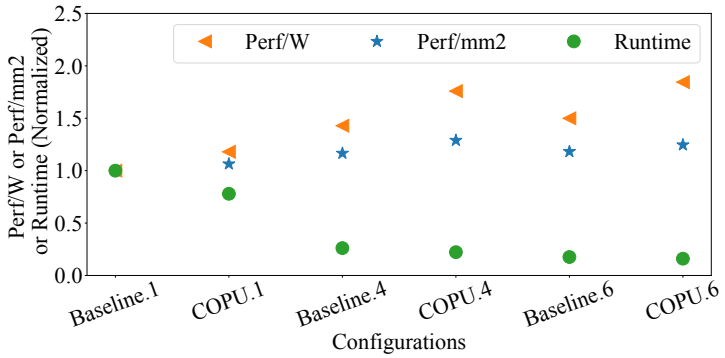


Figure 3.31: Performance/area, performance/power, and latency comparison for different CDU configurations. The baseline represents a CDU without COPU. The suffix number represents the number of OBB-Environment CDUs in Figure 3.21.

energy consumption with 1.13× speedup compared to baseline. As described in Section 3.4.7, the Query Dispatcher proposed in this work prioritizes CDQs that are predicted to be in colliding. This dispatcher schedules a CDQ from `QNONCOLL` only if the queue is full or the predictor has received all CDQs for a given motion. This results in a waiting period before executing any CDQs. The CDU can only run one CDQ at a time for Config-1. In this case, the overhead of the Query Dispatcher's waiting period is less. However, Config-2 has significant parallelism available, and the waiting time overhead becomes prominent. This is expected as the Query Dispatcher prioritize energy efficiency. However, the Query Dispatcher's policy can be designed to find a trade-off between runtime and energy efficiency. Note that collision prediction increases launch-to-execute latency of a CDQ due to collision prediction latency and queuing. However, end-to-end latency of of motion collision detection reduces with collision prediction.

We further compare the runtime and performance with and without collision prediction. Here, the runtime represents average end-to-end latency for motion-environment collision check. Figure 3.31 compares the performance/mm^2, performance/watt, and runtime for different configurations with and without collision prediction. Here, the suffix in the configuration name represents the number of CDUs per COPU (Figure 3.21). For all configurations, the use of collision pre-

diction (COPU.x) provides lower latency, and higher performance/area and performance/watt compared to the baseline (baseline.x). For example, the proposed COPU increases performance/watt and performance/mm^2 by 1.23$\times$ and 1.11$\times$, respectively, while providing 1.18$\times$ speedup.

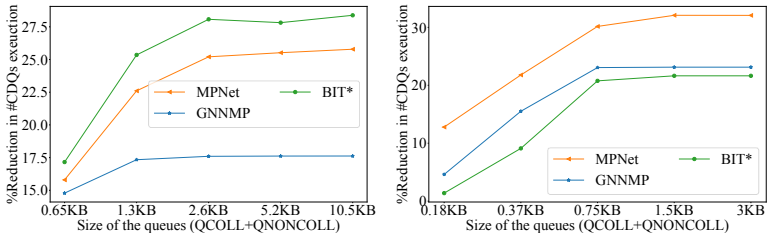
Sensitivity Studies

In this section, we study the impact of Collision Prediction Unit architectural aspects on the reduction achieved in CDQs.

Queue Size: Two queues `QCOLL` and `QNONCOLL` are added in the collision detection unit to store the OBBs generated by the OBB generation unit (Figure 3.21). These queues are added so that the Query Dispatcher can prioritize OBBs predicted to be colliding before performing collision detection for OBBs from the `QNONCOLL` queue. If the `QNONCOLL` queue becomes full before the predictor sends any query to the `QCOLL` queue, then OBBs are dispatched from `QNONCOLL` for collision detection, resulting in redundant computation if the motion is colliding. Thus, the size of `QNONCOLL` queue is crucial and should be set such that it can hold enough entries before the predictor finds a colliding OBB. Figure 3.32 compares the computation reduction achieved by the predictor for different `QNONCOLL` queue sizes. For all benchmarks, the runtime/energy improvement decreases significantly for a very small queue size. The improvement also saturates for larger queue sizes as the queue is not fully utilized during execution.

Collision Prediction Strategy: Figure 3.33a compares the improvement achieved by the proposed predictor for different collision prediction strategies determined by the values of S (Section 3.4.5). As shown in the figure, the improvement is not highly sensitive to the prediction strategy. In several cases, $S = 0$ results in improvement within 2% of the best choice. Using $S = 0$ simplifies the design of the Collision History Table, as only one bit is needed per entry.

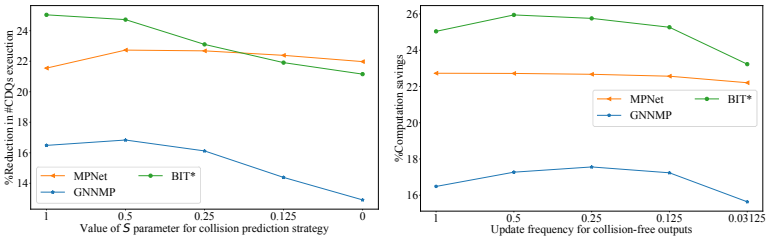
Collision History Table Updates: Figure 3.33b shows the impact of the history table update frequency for collision-free CDQs on the CDQs execution reduction.



(a) Motion planning for 7-DOF robots

(b) 2D motion planning

Figure 3.32: (a) and (b) compares the reduction in CDQs for different size of the QNONCOLL queue.



(a) Collision prediction strategies

(b) Update frequency for collision-free CDQs

Figure 3.33: (a) Compares the CDQs reduction achieved for different collision prediction strategies. (b) Compares the CDQs reduction achieved for different Collision History Table update frequency for collision-free CDQs.

For collision-free queries, we scale the update frequency by $U < 1$ (described in Section 3.4.5). As shown in Figure3.33b, the reduction in the number of executed CDQs is not highly sensitive to the update frequency, this suggests that hash table update frequency can be reduced for reduced traffic.

3.7.4 Area and Power

Table 3.2 summarizes the area and power estimation obtained from the synthesis of RTL implementations of SAS, CECDU, and different configurations of MPAccel. The intersection unit is a major contributor to the total area and power. Further, the total area and power of the Intersection Unit are dominated by fixed-point

Table 3.2: Area and power breakdown for hardware units.

Module	Area (mm^2)	Power (mW)
Scheduler	0.110	60.7
CECDU (with four multi-cycle OOCD)	0.694	215.7
OBB Transformation Unit	0.054	51.6
Octree Traversal Unit	0.029	16.7
Intersection Unit (Multi-cycle)	0.143	24.34
Intersection Unit (Pipelined)	0.251	32.57
MPAccel (Scheduler + 16 CECDUs)		
Config 1: 4 multi-cycle OOCDs/CECDU	11.21	3.51W
Config 2: 4 pipelined OOCDs/CECDU	18.12	4.03W

multipliers ($\sim 85\%$), which can be reduced significantly by employing custom-designed multiplier cells [196]. The critical-path delay for pipelined/multi-cycle OOCD is 1.48ns/2.24ns, and is dominated by multipliers. This delay can be reduced by using optimized 16-bit multipliers' standard cells [196] and/or pipelining.

Collision Prediction Unit A Collision History Table is added for collision prediction. Further, two queues are added in each collision detection unit. We estimate the energy and area overhead of these added components to MPAccel using the OpenRAM Memory compiler. We consider MPAccel configuration with 8 CECDUs, where each CECDU consists of 7 multicycle OOCDs, one COPU, and one OBB transformation unit. Overall, the Collision History Table of size 4096×8 results in 2% and 1.03% area and energy overheads. This overhead can be further reduced to 0.55% and 0.28% using a history table of size 4096×1. `QCOLL` and `QNONCOLL` results in 2.2% and 1.2% area and energy overheads, respectively.

3.7.5 Motion Planning Accelerator (MPAccel)

We further evaluate the runtime for motion planning queries for MPNet algorithm using MPAccel. We use an estimate of 12TOPS for the DNN accelerator, which can be achieved by existing DNN accelerators [83, 279]. Similarly, we set the IO bandwidth to 5GBPS. We estimate the latency of the controller using the number of instructions.

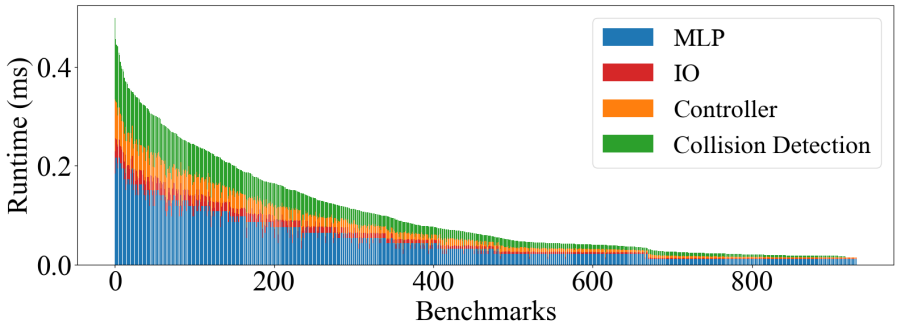


Figure 3.34: Motion planning runtime using MPAccel for different benchmarks. Baxter robot is used for the evaluation. Here the number of CECDUs is set to eight, and each CECDU has four multi-cycle Intersection Units.

Figure 3.34 provides the motion planning latency for different benchmarks for MCSP-based scheduler and 16 CECDUs with four multi-cycle OOCDs each. The motion planning time varies from 0.014ms to 0.49ms, with an average runtime of 0.099ms. This motion planning runtime meets the requirement of real-time motion planning ($< 1ms$ as the actuators' response rate is typically $\sim$ 1kHz).

We further compare the latency and performance of different configurations of the scheduler and CECDUs. Figure 3.35 compares the motion planning latency and performance for different configurations of MPAccel. Here X_Y_mc/p configuration represents MPAccel with X CECDUs, Y OOCDs per CECDU, and multi-cycle (mc) or pipelined (p) design of the Intersection Unit. The left Y-axis compares the latency distribution for motion planning queries. The right Y-axis represents the performance of configurations (motion planning queries/(Second$\times$Watt$\times$mm^2)).

Evaluation on GPU and CPU

Collision detection is the most time-consuming kernel in motion planning [18, 171]. To help compare against GPU baselines, we thus wrote our own OBB-octree collision test for a GPU. Here each thread performs OBB-octree collision detection. We form a warp (32 threads) such that all OBBs in a single warp have physical locality, which reduces divergence as all threads are likely to follow a similar traversal order in the octree. We also implement two optimizations specific to

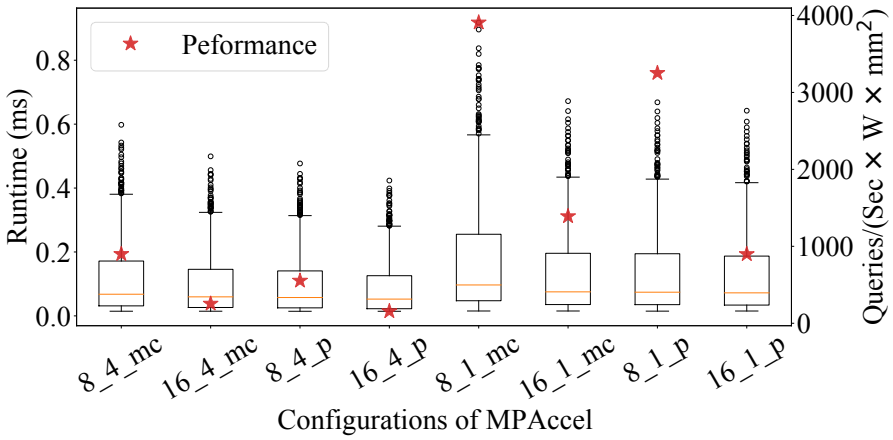


Figure 3.35: Motion planning runtime and performance for different MPAccel configurations. The performance is measured using number of motion planning queries executed per (Second$\times$Watt$\times$mm^2).

Table 3.3: Collision detection runtime for GPU and CPUs.

	NVIDIA Titan V	NVIDIA Jetson TX2 GPU	i7-4771 (8-core)	Cortex-A57 (4-core)
OBB-octree (ms)	24	5833	153	360
OBB-octree + GPU optimizations (ms)	12	3403	N/A	N/A
OBB-octree leaf nodes (ms)	6	1373	890	3304
Power (W)	156.8	3.5	65	4.2
Average motion planning runtime (ms)	1.42	110.27	4.13	11.62

GPU. Each thread accesses a FIFO queue during the octree traversal. We interleave queues for all threads in a single warp to reduce memory divergence. The second optimization is based on [157] to reduce warp divergence. We also implement an OBB-octree leaf nodes collision detection. In this approach, each thread performs collision detection between a leaf node and OBB. Table 3.3 summarizes the CPU and GPU runtime for 2^{20} OBB-octree collision detection queries. For comparison, 16 CECDUs with four multi-cycle OOCDs per CECDU (11.1mm^2, 3.4W) take 0.91ms to execute the same number of OBB-octree collision detection queries.

Similarly, 16 CECDUs with four pipelined OOCDs per CECDU (18.0mm^2, 4.0W) take 0.53ms to execute the same number of OBB-octree collision detection queries. We built a simulator for the CPU+DNN Accelerator and GPU+Controller+DNN Accelerator system to evaluate motion planning runtime. Table 3.3 summarizes the overall motion planning runtime.

3.8 Related Work

Bakhshalipour et al. [11] proposed an acceleration approach for a path planning algorithm for robots with 2-3 DOFs. However, this approach does not apply to motion planning algorithms for robots with higher DOF. They also proposed a collision detection acceleration unit CODAcc based on voxelized OBB-voxelized environment intersection. Here, the number of intersection tests increases with voxelization resolution. We chose the separating axis test as it does not discretize the robot's space and is scalable for fine-resolution intersection tests. Jia et al. [122] proposed a mapping accelerator to map environment point cloud data to octree representation. The proposed accelerator also supports a collision detection test between a voxel and the environment. However, similarly, the number of voxel queries increases with the voxelization resolution used for the robot's occupied space.

Other works have proposed motion planning acceleration for CPUs and GPUs [6, 18, 36, 76, 130, 143, 187, 188, 192]. While GPU-based acceleration approaches provide significant acceleration, state-of-the-art motion planning approaches on GPUs still do not provide the energy efficiency and performance. Hardware acceleration approaches have been proposed for specific motion planning algorithms to meet the computation and real-time requirements, including on FPGA [8, 171, 226] and ASIC [11, 153, 173, 233, 278]. Murray et al. [171] proposed an acceleration approach using FPGAs for probabilistic roadmap-based motion planning (PRM). Further, they expanded the work to a programmable motion planning chip [173, 233]. They proposed to use a fixed set of motions and precompute the space occupied by these motions. These swept spaces are represented using sets of voxels. At runtime, these precomputed swept spaces are used for collision detection. Lian et al. [153] proposed to use octree representation for swept spaces. Though the pre-computation

step reduces the motion planning runtime, to solve challenging motion planning tasks, precomputed swept spaces require more than 40MB on-chip memory or $>$ 40GBPS off-chip memory bandwidth. Yang et al. [278] proposed to use near-memory computing to reduce the memory bandwidth requirement. However, these approaches use a fixed set of motions for motion planning. Such approaches are not suitable for motion planning algorithms where candidate motions are generated dynamically.

RoboRun [24] proposed to control the volume and precision of the environment depending upon its speed and the distance from obstacles. The proposed optimization can be applied to MPAccel, as the environment's octree representation supports variable precision using octree node pruning.

3.9 Conclusion and Future Work

This work focused on accelerating sampling-based motion planning algorithms for autonomous robots. Our key insight behind the proposed approaches is that parallelizing these motion planning algorithms comes at the cost of increased computation and energy consumption per motion planning query. We identify the sources of redundant computations in coarse-grained and fine-grained parallelization. Based on our analysis, we propose algorithm-hardware-based approaches to increase the energy efficiency of parallel execution. SAS, the proposed scheduler unit, improves work-efficiency of inter-pose parallelization results in $1.89\times$ speedup with 41.2% lower energy consumption compared to naive parallelization. Our proposed collision prediction approach, COORD, focuses on inter-CDQ parallelization and provides $17.2\% - 32.1\%$ reduction in the computation on average, increasing performance/Watt by $1.23\times$. The proposed CECDU focuses on accelerating a single CDQ and reduces redundant computation by up to $3\times$. We evaluate MPAccel, a motion planning accelerator consisting of SAS and CECDU. MPAccel enables real-time motion planning for a 7-DOF robot in 0.014ms-0.49ms with an average of 0.099ms when evaluated for a learning-based motion planning algorithm. *Overall, energy-efficient acceleration approaches proposed in this work can be used to push the motion planning capability that can be supported within fixed energy and/or runtime budget, allowing the use of more sophisticated and complex motion*

planning algorithms for high-DOF robots, enabling autonomous robots to perform complex tasks in highly dynamic environments.

We observe that after applying hardware acceleration of collision detection using the proposed approaches, more than 50% of the time is spent on neural motion planning. The next chapter focuses on improving regression neural networks and demonstrates its use for accelerating neural motion planning.

3.9.1 Future Research Directions

Robot collision detection is an exciting workload with several design parameters (e.g., geometric representations of robot and environment occupancy) that affect its computing and memory requirements. Further, the geometric representation of environment occupancy affects the compute/memory overheads for both the robot perception and planning modules. In this work, we assume a black-box perception module that provides an octree representation of the environment occupancy. However, the co-design of geometric representation for perception and planning modules and fine-grained integration of both modules present interesting optimization avenues.

This work focused on discrete collision detection for robot motions and exploited the physical spatial locality and inter-CDQ parallelization for acceleration. However, for collision-free motions, this results in executing all CDQs and concluding an outcome. It becomes the primary source of compute overhead after integrating the proposed optimization. Several approaches have been proposed to calculate swept spaces for motions, reducing the number of CDQs that need to be executed. However, this calculation becomes complicated for high-DOF robots. It is an interesting direction to explore the use of parallelization and physical spatial locality for robot motion-environment collision detection while minimizing the computing for collision-free cases.

This work makes several contributions towards accelerating collision detection for autonomous robots using physical spatial locality. Adapting insights made in this work for accelerating collision detection in other application-domains, such as AR/VR, physics simulation, and games, present possible future directions. Further, while we mainly take advantage of the physical spatial locality to reduce redundant

computation, there is untapped potential for using temporal locality for further reduction.

Robotics workloads have distinct characteristics as several compute tasks are related to the physical space and bodies, resulting in robot- and environment-dependent compute patterns. There is a huge scope for exploiting these application-domain characteristics and application-hardware co-design for building better compute platforms for robotics, as we demonstrated in this work using the physical spatial locality. It would be interesting to apply/build upon observations made in this work for other computing tasks in the robotics pipeline, such as mapping, localization, trajectory optimization, and control.

Chapter 4

Label Encodings for Deep Regression

Deep regression networks, in which a continuous output is predicted for a given input, are traditionally trained by minimizing squared/absolute error of output labels, which we refer to as *direct regression*. However, there is a significant gap in accuracy between direct regression and recent task-specialized approaches for regression problems including head pose estimation, age estimation, and facial landmark estimation. Given the increasing importance of deep regression networks, developing generic approaches to improving their accuracy is desirable.

A regression problem can be posed as a set of binary classification problems using binary-encoded labels. A similar approach has been applied to other domains such as ordinal regression [31, 149] and multiclass classification [4, 45, 54]. Such a formulation allows the use of well-studied binary classification approaches. Further, new generalization bounds for ordinal regression or multiclass classification can be derived from the known generalization bounds of binary classification. This reduces the efforts for design, implementation, and theoretical analysis significantly [149]. Dietterich and Bakiri [54] demonstrated that posing multiclass classification as a set of binary classification problems can increase error tolerance and improve accuracy. However, the proposed approaches for multiclass classification do not apply to regression due to the differences in task objective and properties of the classifiers' error probability distribution (Section 4.3.1). On the other hand, prior works on

ordinal regression have explored the application of binary classifiers in a more restricted way, which limits its application to a wide range of complex regression problems (Section 4.9). *There exists a lack of a generic framework that unifies possible formulations for using binary classification to solve regression.*

In this work, we propose a generic framework for regression by binary classification using *binary-encoded labels (*BEL*)*. BEL improves accuracy of regression networks by generalizing the application of binary classification to regression. In BEL, a target label is quantized and converted to a binary code of length M, and M binary classifiers are then used to learn these binary-encoded labels. An encoding function is introduced to convert the target label to a binary code, and a decoding function is introduced to decode the output of binary classifiers to a real-valued prediction. BEL allows using an adjustable number of binary classifiers depending upon the quantization, encoding, and decoding functions. We propose a new taxonomy for the design space of regression by binary classification and introduce new design aspects, such as quantization, encoding, decoding, and loss functions. BEL opens possible avenues to improve the accuracy of regression problems with a large design space spanning quantization, encoding, decoding, and loss functions.

In this work, we first explore this design space using manually designed functions. We analyze different design aspects and study properties of suitable encoding, decoding, and loss functions for regression problems. First, we focus on the encoding and decoding functions and theoretically study the relations between the absolute error of label and binary classifiers' errors for sample encoding and decoding functions. This analysis demonstrates the impact of binary classifiers' error distribution over the numeric range of target labels on the suitability of different encoding and decoding functions. Based on our analysis and empirically observed binary classifiers' error distribution, we propose properties of suitable encoding functions for regression and explore various manually designed encoding functions on a wide range of tasks. Further, we propose several encoding-specific and generic decoding functions. We also propose an expected correlation-based decoding function for regression that can effectively reduce the quantization error introduced by the use of classification. Further, we analyze the suitability of different loss functions for the proposed framework, including binary cross-entropy, cross-entropy, and squared/absolute error. In this chapter, BEL refers to the use of manually-designed

encoding functions.

Further, we expand on BEL by designing an end-to-end training approach to learn label encodings with network parameters for a given problem. Our evaluation using manually designed encodings demonstrate the importance of finding a suitable encoding for a given benchmark, as different encoding functions result in the lowest error across different problems, network architectures, and datasets. Finding suitable label encoding for a given problem is challenging due to the vast design space. Different approaches, including autoencoder, random search, and simulated annealing, have been proposed to design suitable encoding for multiclass classification [45, 54, 229]. However, these encodings perform relatively poorly for regression due to differences in task objectives (Section 4.6.2). In this work, we propose *Regularized Label Encoding Learning (*RLEL*)*, an end-to-end approach to train the network and label encoding together. RLEL relaxes the assumption of using discrete search space for label encodings. Label encoding design can be approached by regularized search through a continuous space of real-valued label encodings, enabling the use of continuous optimization approaches. Such a formulation enables end-to-end learning of the network parameters and label encoding. We propose two regularization functions to encourage properties identified as being helpful for binary-valued label encodings in the learned label encoding during training.

A deep regression network consists of a feature extractor and a regressor and is trained end-to-end. A regressor is typically the last fully connected layer with one output logit for direct regression. Our proposed regression approach (BEL) can be combined with off-the-shelf task-specific feature extractors by increasing the regressor's output logits. Further, we find that the correlation between multiple binary classifiers' outputs can be exploited to reduce the size of the feature vector and consequently reduce the number of parameters in the regressor.

We evaluate the proposed approach on four complex regression problems: head pose estimation, facial landmark detection, age estimation, and end-to-end learning for self-driving car steering. We further evaluate the impact of model compression approaches on direct regression and label encoding-based regression. We also evaluate the use of binary-encoded labels for a neural path planner network and explore its potential for reducing computation in path planning using smaller and sparser models.

We make the following contributions in this work:

- We propose binary-encoded labels for regression and introduce a general framework and a taxonomy for the design aspects of regression by binary classification.
- We propose desirable properties of encoding and decoding functions suitable for regression problems. We present a series of suitable encoding, decoding, and loss functions for regression with BEL.
- We provide a label encoding design approach by combining regularizers with continuous search of label encodings. We analyze properties of suitable encodings in the continuous search space and propose regularization functions for end-to-end learning of network parameters and label encoding.
- We combine BEL with task-specific feature extractors for four regression problems and evaluate multiple manually designed encoding, decoding, and loss functions. We also evaluate the proposed label encoding learning approach (RLEL) for all benchmarks. BEL and RLEL outperform direct regression for all the problems and specialized approaches for several tasks.
- We study the impact of sparsity on regression error for direct regression, BEL, and RLEL for several benchmarks. Label encoding-based regression enables the use of smaller and sparser models with less accuracy degradation compared to direct regression for several computer vision and robot path planning tasks.

We first provide details of the proposed framework and taxonomy of regression by binary classification in Section 4.1. Next, we provide an analytical study to compare different encoding and decoding functions to motivate the exploration of this design space in Section 4.2, which also motivates proposed design properties of suitable encoding and decoding functions. Section 4.3 provides details of the proposed properties of suitable encoding, decoding, and loss functions and introduce several manually designed functions. The network architecture used for manually designed functions is introduce in Section 4.3.4. Section 4.4 provides details of the proposed approach for label encoding learning, regularization functions formulation,

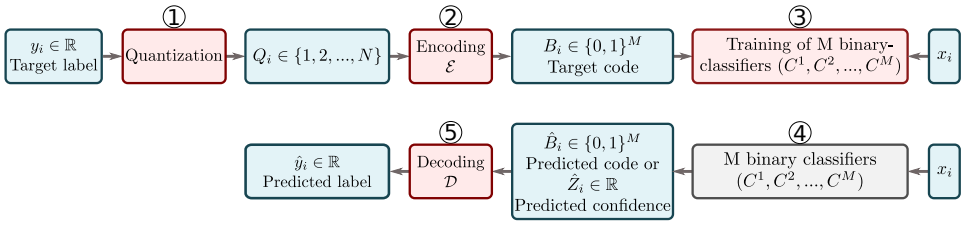


Figure 4.1: The training (top) and inference (bottom) flow of binary-encoded labels (BEL) for regression networks. Red colored blocks represent design aspects we focus on.

and network architecture. Section 4.5 and Section 4.6 provides the experimental methodology and evaluation for manually designed and learned label encodings, and compare with existing approaches. Further, we provide two case studies to demonstrate the use of label encodings to reduce inference computation/energy for computer vision and robot path planning tasks in Section 4.7 and Section 4.8.

4.1 Regression by Binary Classification

We consider regression problems where the goal is to minimize the error between real-valued target labels y_i and predicted labels $\hat{y}_i$, over a set of training samples i. We transform this problem to a set of binary classification sub-problems by converting a real-valued label to a binary code.

Figure 4.1 shows the training and inference flow for BEL. The red-colored blocks highlight functions that vary under BEL. A real-valued label $y_i \in \mathbb{R}$ is quantized to level $Q_i \in \{1, 2, ..., N\}$ ①. The quantized label is converted to a binary vector $B_i \in \{0,1\}^M$, that we call a *binary-encoded label*, using encoding function $\mathscr{E}$ ②. There are $\binom{2^M}{N}$ possible encoding functions—a large number. The binary-encoded labels B_i are used to train M classifiers ③. During inference the M classifiers predict a binary code $\hat{B}_i \in \{0,1\}^M$ for input x_i ④. The predicted code ($\hat{B}_i$) or the predictions' magnitude ($\hat{Z}_i$), which indicates its confidence, is then decoded to a predicted label $\hat{y}_i \in \mathbb{R}$ using a decoding function $\mathscr{D}$ ⑤. We explore decoding functions that yield either quantized or continuous predicted outputs. The latter avoids quantization error by employing expected correlation (Section 4.3.2). Table 4.1 summarizes the notations used in this work.

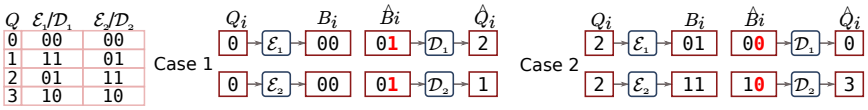


Figure 4.2: Examples of two lookup table-based encoding and decoding functions. $\hat{Q}_i$ represents the decoded quantized level.

Table 4.1: Summary of notations used in this work

Notation	Description
x_i, y_i, Q_i	Input, real-valued target label, and quantized target label for training example i. $y_i \in [1,N]$ and $Q_i \in \{1,2,...,N\}$
N	The range of target labels y_i; Number of quantization levels for Q_i
M	Number of bits/values for label encoding
B_i, $\hat{B}_i$	Target and predicted binary-encoded labels (used for hand-crafted label encoding
$\hat{Z}_i$	Predicted real-valued encodings; activation values of the output code layer in Figure 4.8
$\mathscr{E}$	Encoding function; $B_i = \mathscr{E}(Q_i)$
$\mathscr{D}$	Decoding function; $\hat{y}_i = \mathscr{D}(\hat{Z}_i)$
E	Learned label encoding through RLEL; calculated from $\hat{Z}_i$ for all training examples using equation 4.7
$\hat{C}_i$	Output correlation vector of length N. Here $\hat{C}_i^j$ gives a measure of the probability that predicted label value is equal to j
D	Decoding matrix that converts the predicted encodings to a correlation vector $\hat{C}_i$

Figure 4.2 illustrates how encoding and decoding functions $\{\mathscr{E},\mathscr{D}\}$ can impact accuracy. The figure shows two distinct lookup table based encoding/decoding functions $\{\mathscr{E}_1,\mathscr{D}_1\}$ and $\{\mathscr{E}_2,\mathscr{D}_2\}$. Case 1 and 2 illustrate examples of encoding a given target value and decoding an erroneous predicted code (italicized red bits indicate misclassification). For a given classification error, $\{\mathscr{E}_2,\mathscr{D}_2\}$ exhibits lower regression error versus $\{\mathscr{E}_1,\mathscr{D}_1\}$.

The proposed framework contains five major design parameters resulting in a large design space: quantization, encoding, decoding, regressor network architecture, and training loss formulation. Section 4.3 explores the characteristics of suitable encoding, decoding, and loss functions. In this work we consider only uniform quantization while leaving nonuniform quantization [72] to future work. We find varying any of these aspects can improve accuracy. While BEL provides a framework, and some design choices appear generally better than others, the most suitable BEL parameters to employ vary across task, dataset, and network architecture, as we show both theoretically (Section 4.2) and empirically (Section 4.6).

4.2 Analytical Comparison of Encoding and Decoding Functions

This section analyzes the potential impact of encoding/decoding functions on regression error assuming empirically observed error distributions for the underlying classifiers. We compare Unary and Johnson codes (Figure 4.3b and 4.3c) to determine when each is preferable. The significance of these particular codes and their use for regression problems is explained in Section 4.3. With this analytical study, we aim to obtain insight into ordinal label classifier's impact on regression error when employing simple encoding and decoding functions $\{\mathscr{E},\mathscr{D}\}$. Based upon this analysis we identify desirable properties for these functions. The design of the codes and intuition for trying them are discussed in Section 4.3.1. We divide our analysis into three parts: First, the expected error of predicted labels is derived in terms of classifiers' errors for two $\{\mathscr{E},\mathscr{D}\}$ functions. Next, we propose an approximate classifier's error probability distribution over the numeric range of target labels for regression based on empirical study. Last, we compare the expected error of sample $\{\mathscr{E},\mathscr{D}\}$ functions based on our analysis. We use labels $y_i \in [1, N-1]$, with quantization levels $Q_i \in \{1,2,...,N-1\}$. Quantization error is not included as it is not affected by $\{\mathscr{E},\mathscr{D}\}$ functions.

Expected absolute error bounds in terms of classification error: First, we analyze the unary code (BEL-U). The encoding function $\mathscr{E}^{\text{BEL-U}}$ converts Q_i to $B_i = b_i^1, b_i^2, ..., b_i^{N-2}$, where $b_i^k = 1$ for $k < Q_i$, else 0. In this case, a good choice of decoding function turns out to be simply counting the number of 1 outputs across all $N-2$ classifiers since a error in a single classifier changes the prediction by only one quantization level. Adding one since $Q_i = 1$ is encoded by all zeros gives:

$$\mathscr{D}^{\text{BEL-U}}(\hat{b}_i^1, \hat{b}_i^2, ..., \hat{b}_i^{N-2}) = \sum_{k=1}^{N-2} \hat{b}_i^k + 1 \tag{4.1}$$

Let $e_k(n)$ be the error probability of classifier k for target quantized label $Q_i = n$. For a uniform distribution of y_i in the range $[1, N-1]$ the expected error for BEL-U

Q_i	$b^1b^2b^3b^4b^5b^6b^7$
1	0 0 0 0 0 0 0
2	1 0 0 0 0 0 0
3	1 1 0 0 0 0 0
4	1 1 1 0 0 0 0
5	1 1 1 1 0 0 0
6	1 1 1 1 1 0 0
7	1 1 1 1 1 1 0
8	1 1 1 1 1 1 1

(a) **(b)** Unary

$b^1b^2b^3b^4$
0 0 0 0
0 0 0 1
0 0 1 1
0 1 1 1
1 1 1 1
1 1 1 0
1 1 0 0
1 0 0 0

(c) Johnson

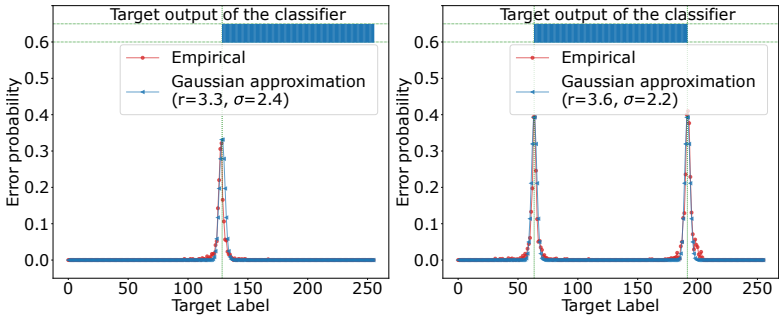
Figure 4.3: Examples of BEL codes. Part (a) represents the quantized values of the labels for Unary and Johnson codes shown in Parts (b) and (c).

can be shown (see Appendix A.1) to be bounded as follows:

$$\mathbb{E}(|\hat{y}^{\text{BEL-U}} - y|) \leqslant \frac{1}{N-1} \sum_{n=1}^{N-1} \Big(\sum_{k=1}^{N-2} e_k(n) \Big) \tag{4.2}$$

A similar analysis of expected error can be applied to binary encoded labels constructed to yield Johnson codes (BEL-J), in which Q_i is encoded using $B_i = b_i^1, b_i^2, ..., b_i^{N/2}$, where, $b_i^k = 1$ for $\frac{N}{2} - Q_i < k - 1 \leqslant N - Q_i$, else 0 (see Equation A.21 in Appendix A.1) .

Error probability of classifiers: To use Equation 4.2 we need to determine $e_k(n)$. A classifier's target output is 0 or 1. For BEL, the target labels of a given classifier will have one or more *bit transitions* from $0 \rightarrow 1$ or $1 \rightarrow 0$ as the target value of the regression network's output varies. For example, for the unary code (Figure 4.3b), the target output of classifier b^2 has a bit transition from 0 to 1 going from $Q_i = 2$ to $Q_i = 3$. The classifier should learn a decision boundary in $(2,3)$. Each BEL classifier is tasked with learning decision boundaries for all bit transitions. As the difficulty of this task varies with the number of bit transitions it varies with different encoding functions. Moreover, the misclassification probability of a classifier tends to increase as the target label is closer to the classifier's decision boundaries [35]. Thus, we approximate $e_k(y)$ for a classifier k with t bit transitions as a linear combination of t Gaussian distributions. Here, each Gaussian term is centered around a bit transition. Let $f_{\mathcal{N}(\mu,\sigma^2)}(y)$ denote the probability density of a normal distribution with mean μ and variance σ^2. Each classifier for BEL-U encoding has one bit transition, whereas,



(a) Classifier A **(b)** Classifier B

Figure 4.4: Part (a) and (b): classification error probability vs. target output for two classifiers. Target output 1 where blue and 0 elsewhere.

each classifier for BEL-J encoding has two bit transitions (except the first and last classifiers). $e_k(y)$ of a classifier k for BEL-U and BEL-J encoding is approximated as:

$$e_k^{\text{BEL-U}}(y) = rf_{\mathcal{N}(\mu_k,\sigma^2)}(y), \text{where, } \mu_k = k + 0.5 \tag{4.3}$$

$$e_k^{\text{BEL-J}}(y) = rf_{\mathcal{N}(\mu_{1k},\sigma^2)}(y) + rf_{\mathcal{N}(\mu_{2k},\sigma^2)}(y), \text{where, } \mu_{1k} = \frac{N}{2} - k + 1.5, \mu_{2k} = N - k + 1.5 \tag{4.4}$$

Here, r is a scaling factor. Figure 4.4a and 4.4b compares Equation 4.3 and 4.4 against empirically observed error distributions for two classifiers using an HRNetV2-W18 [264] feature extractor (backbone) trained with the COFW facial landmark detection dataset [29].

Comparison of expected absolute error for BEL-U and BEL-J: Based on the above analysis, we compare the expected absolute errors of BEL-U and BEL-J. Figure 4.5 represents the percentage increase in absolute error for BEL-U compared to BEL-J for valid values of standard deviation σ ($y-$axis) and scaling factor r ($x-$axis) as used in Equation 4.3 and 4.4. Here, BEL-J has a lower error in the red-colored region (%increase> 0), whereas BEL-U has a lower error in the blue-colored region (%increase < 0). The figure shows that whether BEL-J or BEL-U has lower error depends upon the values of σ and r. This suggests that the best $\{\mathscr{E}, \mathscr{D}\}$ function will depend upon the classifier error probability distribution. The classifier error distribution in turn may depend upon the task, dataset, label distribution, network

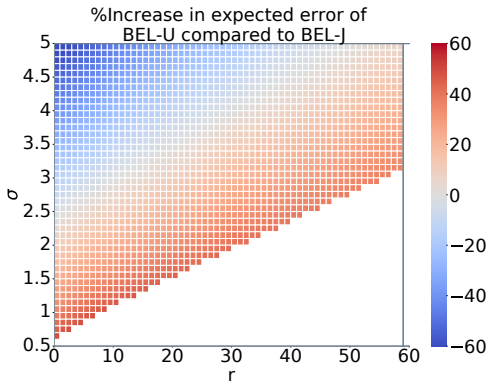


Figure 4.5: Expected error increase of BEL-U versus BEL-J based on Equation 4.2 to Equation 4.4 (blank means that combination of r and σ results in an error probability greater than one).

architecture, and optimization approach.

4.3 Binary-Encoded Label Design

In this section, we discuss the properties of suitable design aspects in BEL, and propose several manually designed functions based on these proposed properties.

4.3.1 Design of Encoding Functions

Based on the above analysis and further empirical observation we identify three principles for selecting BEL codes for regression so as to minimize error. First, *individual classifiers should require fewer bit transitions* as this makes them easier to train. Second, a desirable property for a BEL encoding function is that *the hamming distance between two codes (number of bits that differ) should be proportional to the difference between the target values they encode.* However, hamming distance weighs all bit changes equally. Thus, hamming distance based code design provides equal error protection capability to all bits [270, 273] and does not account for which classifiers are more likely to mispredict for a given input. This matters because the misclassification probability of BEL classifiers is not uniform, but rather increases the closer the target value of an input is to a bit transition (e.g., Figure 4.4a and 4.4b). These observations yield a third important consideration: *For a given target*

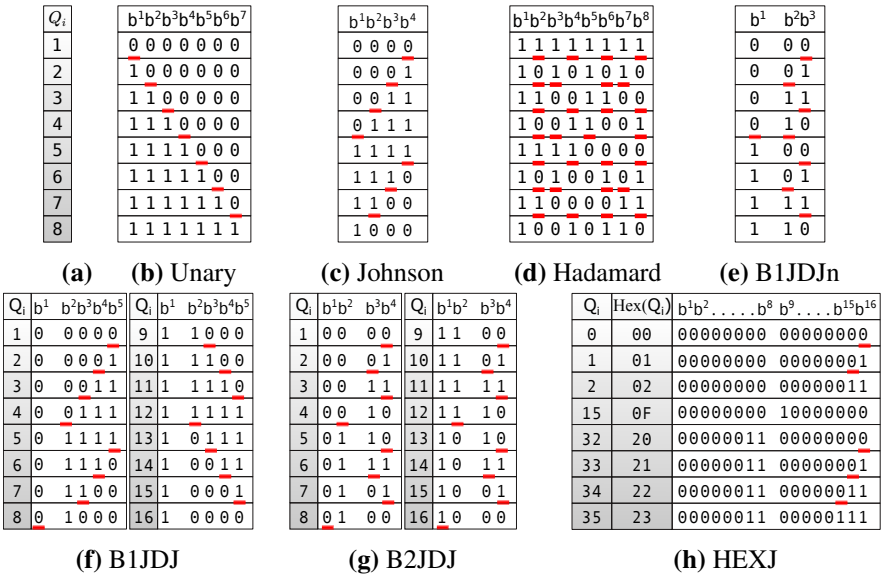
value classifiers closer to a bit transition are more likely to incur an error.

The principles above highlight a trade-off between classification error probability and error-correction properties when selecting BEL codes. As mentioned before, the hamming distance between two codes gives a measure of the error correction capability. Error-correcting codes such as Hadamard code provides high hamming distance between two codes. However, for Hadamard codes in regression by binary classification, many classifiers (i..e, bit positions) have several bit transitions, as shown in Figure 4.6d, which increases the classification error. We empirically observe that classifier for b^7 has classification error of 2% for Unary code and 50% for Hadamard code. Thus, a trade-off exists between the number of bit transitions that can be learned by the network for a given problem and the error correction provided by the used label encodings. Development of algorithms that might systematically optimize encoding functions is explored in the next Section. To evaluate the trade-offs, we empirically evaluate encodings that, to greater or lesser extent, satisfy one or more of the principles while focusing on reducing the number of classifiers (bits) so as to avoid increasing model parameters. Specifically, we explore the following codes:

Unary code (U): Unary codes (Figure 4.6b) have only one bit transition per classifier and thus require $M = N - 1$ bits to encode N values. Unary codes satisfy the first two principles and prior work on ordinal regression by binary classification [149, 177] uses similar codes.

Johnson code (J): The Johnson code sequence (Figure 4.6c) is based on Libaw-Craig code [155]. We select this code as it has well-separated bit transitions and requires $M = \frac{N}{2}$ bits compared to N required for unary code. This code exemplifies the impact of considering non-uniform classifier error probabilities (third principle). For example the hamming distance between 1 and 8 is just one. However, the bit transition for the differing bit, for classifier b^1, is far from 1 or 8. Assuming equal error probability distributions centered on each bit transition for each classifier (as in Equation 4.4), b^1 is less likely to mispredict than b^2, b^3 or b^4 for inputs with target values near 1 or 8. Thus, even though only one bit differs between codes for labels 1 and 8, this bit is less likely to be erroneous, providing higher effective distance between these two codes.

Base+displacement based code (B1JDJ/B2JDJ): We further reduce the num-



Q_i	$b^1b^2b^3b^4b^5b^6b^7$
1	0 0 0 0 0 0 0
2	1 0 0 0 0 0 0
3	1 1 0 0 0 0 0
4	1 1 1 0 0 0 0
5	1 1 1 1 0 0 0
6	1 1 1 1 1 0 0
7	1 1 1 1 1 1 0
8	1 1 1 1 1 1 1

(a) **(b)** Unary

$b^1b^2b^3b^4$
0 0 0 0
0 0 0 1
0 0 1 1
0 1 1 1
1 1 1 1
1 1 1 0
1 1 0 0
1 0 0 0

(c) Johnson

$b^1b^2b^3b^4b^5b^6b^7b^8$
1 1 1 1 1 1 1 1
1 0 1 0 1 0 1 0
1 1 0 0 1 1 0 0
1 0 0 1 1 0 0 1
1 1 1 1 0 0 0 0
1 0 1 0 0 1 0 1
1 1 0 0 0 0 1 1
1 0 0 1 0 1 1 0

(d) Hadamard

b^1	b^2b^3
0	0 0
0	0 1
0	1 1
0	1 0
1	0 0
1	0 1
1	1 1
1	1 0

(e) B1JDJn

Q_i	b^1	$b^2b^3b^4b^5$
1	0	0 0 0 0
2	0	0 0 0 1
3	0	0 0 1 1
4	0	0 1 1 1
5	0	1 1 1 1
6	0	1 1 1 0
7	0	1 1 0 0
8	0	1 0 0 0

Q_i	b^1	$b^2b^3b^4b^5$
9	1	1 0 0 0
10	1	1 1 0 0
11	1	1 1 1 0
12	1	1 1 1 1
13	1	0 1 1 1
14	1	0 0 1 1
15	1	0 0 0 1
16	1	0 0 0 0

(f) B1JDJ

Q_i	b^1b^2	b^3b^4
1	0 0	0 0
2	0 0	0 1
3	0 0	1 1
4	0 0	1 0
5	0 1	1 0
6	0 1	1 1
7	0 1	0 1
8	0 1	0 0

Q_i	b^1b^2	b^3b^4
9	1 1	0 0
10	1 1	0 1
11	1 1	1 1
12	1 1	1 0
13	1 0	1 0
14	1 0	1 1
15	1 0	0 1
16	1 0	0 0

(g) B2JDJ

Q_i	Hex(Q_i)	$b^1b^2.....b^8$ $b^9....b^{15}b^{16}$
0	00	00000000 00000000
1	01	00000000 00000001
2	02	00000000 00000011
15	0F	00000000 10000000
32	20	00000011 00000000
33	21	00000011 00000001
34	22	00000011 00000011
35	23	00000011 00000111

(h) HEXJ

Figure 4.6: Examples of BEL codes. Part (a) represents the quantized values of the labels for Unary and Johnson codes shown in Parts (b) and (c). Part (d) shows a B1JDJ code without reflected binary; Parts (e) and (f) show B1JDJ and B2JDJ codes for targets in the range 1 to 16. Part (g) shows quantized and encoded values for a HEXJ code (space added to differentiate between base and displacement, or digits). Red lines represent bit transitions. These BEL codes described in Section 4.3.1.

ber of bits using a base+displacement-based representation. In this representation, a value is represented in base-k using a base-term `b` and displacement `d` via `b` * k + `d`. `b` and `d` are represented using Johnson codes. Further, to improve the distance between two remote codes, we adapt reflected binary codes for term `d` [87]. We evaluate base-2 (B1JDJ - Figure 4.6f) and base-4 codes (B2JDJ - Figure 4.6g).

Binary coded hex - Johnson code (HEXJ): In HEXJ (Figure 4.6h), each digit (0-F) of the hexadecimal representation of a number is converted to an 8-bit binary code using Johnson code. For example, for the decimal number 47 (i.e., 2F in hex), HEXJ(47) = Concatenate(Johnson(2), Johnson(F)). A 16-bit HEXJ code can represent numbers in the range of 00 to FF (a total of 256). The number of bits increases sub-linearly with the number of quantization levels for HEXJ, making it suitable for regression problems with many quantization levels.

Hadamard code (HAD): Hadamard codes [25] are widely used as error-correcting codes and have been used for multiclass classification [54, 259]. They require $M = N$ bits to encode N values. However, Hadamard codes violate all three BEL code selection principles: First, each classifier has many bit transitions. Second, as each code is equidistant (hamming distance of $\frac{M}{2}$), the difference between target values is ignored. Finally, they protect all bits equally so do not take advantage of non-uniform error probabilities. We verify empirically Hadamard codes are unsuitable for regression.

4.3.2 Design of Decoding Functions

We explore three decoding functions: encoding-specific decoding, correlation-based decoding, and expected correlation-based decoding. Custom decoding functions are specific to the encoding function, and are only evaluated for unary and Johnson codes (described in Section 4.2). In contrast, correlation-based decoding, first explored in prior work studying ECOC for multiclass classification [4], can be applied to all codes. For quantized labels in $\{1, 2, ..., N\}$, we define a code matrix $\boldsymbol{C}$ of size $N \times M$, where M is the number of bits/classifiers used for the binary-encoded label. Each row $\boldsymbol{C}_{k,:}$ in this matrix represents the binary code for label $Q_i = k$. For example, Figure 4.6b can be considered a code matrix, where the first row represents code for label $Q_i = 1$. Let $\hat{Z}_i \in \mathbb{R}^M$ denote the output logit values of the classifiers. For decoding, the row with the highest correlation with the output $\hat{Z}_i$ is selected as the decoded label. Here, real-valued output $\hat{Z}_i$ is used instead of output binary code $\hat{B}_i$ to find the correlation as it uses the confidence of a classifier to make a more accurate prediction. For target quantized labels $Q_i \in \{1, 2, ..., N\}$, the decoding function is defined as:

$$\mathscr{D}^{\text{GEN}}(\hat{Z}_i) = \underset{k \in \{1,2,...,N\}}{\text{argmax}} \left(\hat{Z}_i \cdot \boldsymbol{C}_{k,:} \right) \tag{4.5}$$

However, $\mathscr{D}^{\text{GEN}}$ outputs a quantized prediction, introducing quantization error. To remedy this concern and demonstrate the potential of more sophisticated decoding rules, we propose and evaluate an expected correlation-based decoding function, which allows prediction of real-valued label $\hat{y}_i$. For target labels $y_i \in [1, N]$, the

decoding function is defined as:

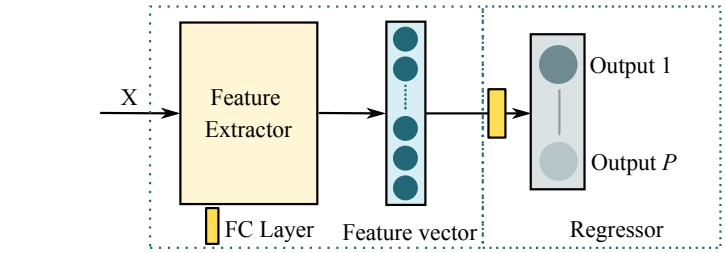
$$\mathscr{D}^{\text{GEN-EX}}(\hat{Z}_i) = \sum_{k=1}^{N} k\sigma_k, \text{where } \sigma_k = \frac{e^{\hat{Z}_i \cdot \boldsymbol{C}_{k,:}}}{\sum_{j=1}^{N} e^{\hat{Z}_i \cdot \boldsymbol{C}_{j,:}}} \tag{4.6}$$

4.3.3 Training Loss Functions

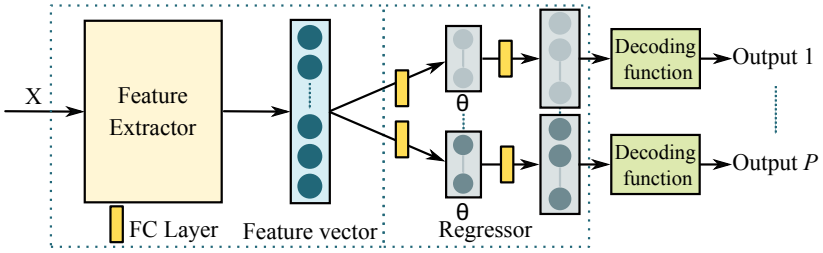
A deep neural network with multiple output binary classifiers can be trained using the binary cross-entropy (BCE) loss $\mathscr{L}_{\text{BCE}}\big(\hat{Z}_i, \mathscr{E}(Q_i)\big)$. However, this loss minimizes the mismatch between predicted and target code but does not directly minimize the error between the target and predicted values. Decoding functions $\mathscr{D}^{\text{GEN}}$ and $\mathscr{D}^{\text{GEN-EX}}$ can be used to calculate the loss and minimize the mismatch between decoded predictions and target values directly. Decoding function $\mathscr{D}^{\text{GEN}}$ finds the correlation between each row of the code matrix ($\boldsymbol{C}_{k,:}$) and the output $\hat{Z}_i$. $\boldsymbol{C}\hat{Z}_i$ gives the correlation vector, and the index with the highest correlation is used as the predicted label. In this case, cross-entropy loss $\mathscr{L}_{\text{CE}}\big(\boldsymbol{C}\hat{Z}_i, Q_i\big)$ can be used to train the network. Similarly, for decoding function $\mathscr{D}^{\text{GEN-EX}}$, which predicts a continuous value, L1 or L2 loss $\mathscr{L}_{\text{L1/L2}}\big(\mathscr{D}^{\text{GEN-EX}}(\hat{Z}_i), y_i\big)$ can also be used for training. We evaluate multiple combinations of decoding and loss functions in Section 4.6.

4.3.4 Network Architecture for BEL

A regression network typically consists of a feature extractor and regressor. The regressor consists of a fully connected layer between the feature extractor's output (i.e., feature vector) and output logits for direct regression as shown in Figure 4.7a. In BEL, the number of output logits is increased to the number of classifiers (bits) used. When $y \in \mathbb{R}^P$, with $P > 1$, the required number of output logits—$P \times M$ assuming M-bit BEL encoding per output dimension—can significantly increase the size of the regression layer. However, empirically, we find small feature vectors suffice as the output logits are highly correlated for the explored encoding functions. Adding a fully connected bottleneck layer to reduce feature vector size to θ reduces the number of parameters and provides a trade-off between the model size and accuracy. Figure 4.7b shows the modified network architecture for BEL.



(a) Direct regression



(b) Binary-encoded labels

Figure 4.7: Network architecture for direct and BEL regression; only the regressor architecture is modified, but the entire network is trained end to end. P is the number of dimensions of the regression network output.

4.4 Regularized Label Encoding Learning

The use of label encodings in regression introduces a large design space. We first analyzed different design aspects of this space and proposed properties of suitable encoding functions. We introduced several manually designed encoding functions based on these proposed properties in the previous section. However, our evaluation shows that the regression error varies significantly across different encoding functions (Section 4.6.1) for a given benchmark. Moreover, encoding function resulting in the least regression error varies across different benchmarks (combination of task, network architecture, and dataset). Thus, and automated approach for finding suitable label encoding function for a given benchmark is desirable. This section explains the proposed label encoding learning approach RLEL.

First, we explain the regression by binary classification formulation used in this work for end-to-end training of network parameters and label encoding. Further, we relax the assumption of binary label encoding and introduce properties of suitable

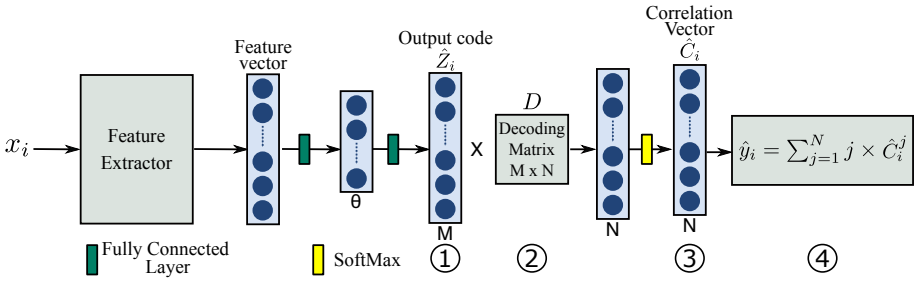


Figure 4.8: The flow for combined training of feature extractor and label encoding.

label encodings in continuous space. Lastly, we explain the proposed regularizers and loss function that accelerate the search for label encodings by encouraging learned label encoding to exhibit the proposed properties.

4.4.1 Label Encoding Learning Framework

Figure 4.8 represents the formulation used in this work for label encoding learning. x_i and y_i represent the input and the real-valued target label for sample i, respectively. We assume $y_i \in [1,N]$ for simplicity as the real-valued targets with any arbitrary numeric range can be scaled and shifted to this range. $Q_i \in \{1,2,...,N\}$ represents the quantized target label. The input x_i is passed through a feature extractor and fully connected (FC) layers to generate the predicted encoding $\hat{Z}_i \in \mathbb{R}^M$ ①. Here, an FC layer of size θ ($\theta < M$) is added between the feature vector and output code. This layer reduces the number of parameters in FC layers and improves accuracy, as we found empirically. Each neuron of the output code is a binary classifier, and the magnitude $\hat{Z}_i^k$ gives a measure of the confidence of the classifier-k [4]. The output code and a decoding matrix $D \in \mathbb{R}^{M \times N}$ are multiplied ②, and the output is passed through a softmax function to give a correlation vector $\hat{C}_i \in \mathbb{R}^N$ ③, where the value of $\hat{C}_i^k$ represents the probability that the predicted label $\hat{y}_i = k$. This correlation vector is then converted to a real-valued prediction by taking the expected value ④.

Manually designed encoding functions proposed in the previous section provides binary label encodings $B_i = \mathscr{E}(Q_i) \in \{0,1\}^M$. The network can be trained using binary cross-entropy loss between B_i and $\hat{Z}_i$, and these encodings are used as columns of the decoding matrix ($D_{:,i} = \mathscr{E}(i)$). However, it is desirable to automati-

cally find suitable encodings B_i and decoding matrix D without searching through a set of hand-designed encodings.

The search space of binary label encodings is discrete and hence challenging to search using traditional continuous optimization methods [49]. Hence, we relax the assumption of binarized label encodings and use a continuous search space. This relaxation, coupled with the proposed formulation, enables the use of traditional optimizers to learn label encoding and the decoding matrix D with the entire network by optimizing the loss between targets and prediction. Let $\mathbb{S}_n$ represent the set of training samples with quantized target $Q_i = n$, and $E \in \mathbb{R}^{N \times M}$ represent a label encoding matrix, where each row $E_{n,:}$ is the encoding for target $Q_i = n$. E is defined as:

$$E_{n,:} = \frac{1}{|\mathbb{S}_n|} \sum_{i \subset \mathbb{S}_n} \hat{Z}_i \tag{4.7}$$

Training the network solely with the loss between $\hat{y}_i$ and y_i does not constrain the search space of label encodings (E). In regression, the label encoding (E) significantly impacts the accuracy, and label encodings that follow specific properties result in lower error(Section 4.6.1). The following section explains desirable characteristics of output codes for regression and how these properties can be encouraged in learned label encoding using regularization functions.

4.4.2 Label Encoding Learning with Regularizers

Section 4.3 summarizes the properties of suitable binary label encodings for regression and introduces manually designed encoding Functions. These properties constrain the vast search space of binary label encodings. We further expand on these properties to make it suitable for real-valued label encodings. Each of these properties is used to design a regularization function that can be combined with training loss. These regularization functions encourage the learned label encodings to follow the proposed properties, narrowing the search space of label encodings during training.

We propose two regularizers applicable to learned label encoding (E) to limit its search space. E is measured from the output codes $\hat{Z}_i$ over the complete training dataset (Equation 4.7). However, deep neural networks are trained using mini-batches, where each batch consists of K training examples sampled randomly from

the (typically shuffled) training set. We extend the proposed regularization rules to apply to a minibatch-based loss function.

Regularizer R1 - Distance between encodings:

A binary classifier's real-valued output represents its confidence (i.e., error probability). The L1 distance between real-valued predicted encodings gives more weight to classifiers that are more confident (i.e., higher value). Thus, by considering the L1 distance between real-valued codes instead of the hamming distance between binary codes, we can combine the second and third design properties of binary label encodings (Section 4.3.1) into a single rule for real-valued label encodings. This gives the first regularization rule: *L1 distance between encodings for two labels should increase with the difference between two labels*, i.e., $||E_{i,:} - E_{j,:}||_1 \propto |i-j|$.

Regularization Function Derivation

The proposed property suggests $||E_{i,:} - E_{j,:}||_1 \propto |i-j|$.

So ideally, $||E_{i,:} - E_{j,:}||_1 = \lambda |i-j|$

Since $E_{x,:}$ is average of $\hat{Z}_i$ for samples with label value x (equation 4.7), the above condition leads to:

$$||\hat{Z}_i - \hat{Z}_j||_1 = \lambda |y_i - y_j| \tag{4.8}$$

Based on this requirement, we add the following regularization function:

$$\text{Regularization Loss} = \max(0, \lambda |y_i - y_j| - ||\hat{Z}_i - \hat{Z}_j||_1) \tag{4.9}$$

The above loss term penalizes the encodings if $||\hat{Z}_i - \hat{Z}_j||_1 < \lambda |y_i - y_j|$. It does not strictly impose equation 4.8. However, it approximately imposes the constraint as per shown in empirical verification in Section 4.6.4. Further, we set the scaling parameter λ to 2 based on binary-encoded labels. For two adjacent labels (i.e., $|y_i - y_j| = 1$), the loss function encourages $||\hat{Z}_i - \hat{Z}_j||_1$ to be greater than 2. Here, $\hat{Z}$ is the output encodings. In the case of binarized label encoding (-1 if $Z < 0$ and $+1$ if $Z > 0$), $||Z_i - Z_j||_1 = 2$ signifies that two encodings differ in at least one bit.

Thus, Regularizer R1 can be approximated as the following for a batch with K training examples:

$$\mathscr{L}_1 = \sum_{i=1}^{K} \sum_{j=1}^{K} \max(0, 2 \times |y_i - y_j| - ||\hat{Z}_i - \hat{Z}_j||_1) \tag{4.10}$$

The above regularization considers K^2 pairs in a minibatch of K samples, and impose the regularization rule from Equation equation 4.9 on all pairs in the given minibatch.

Regularizer R2 - Regularizing bit transitions:

The number of bit transitions in a bit-position of label encoding gives a measure of the binary classifier's decision boundary's complexity. Thus the number of bit transitions has to be chosen for a given problem based on the trade-off between classification error and error-correction capability. This trade-off can be found using by regularizing the number of bit-transitions in the learned label encoding. However, there are no $0 \rightarrow 1$ or $1 \rightarrow 0$ transitions in real-valued label encodings. Thus, we approximate the number of bit transitions by measuring the L1 distance between encodings for adjacent label values $Q_i = n$ and $Q_i = n+1$. The number of bit transitions for real-valued label encoding E can be approximated as:

$$\sum_{i=1}^{M} \sum_{j=1}^{N-1} |E_{j,i} - E_{j+1,i}| \tag{4.11}$$

This leads to the second regularization rule: *The L1 distance between encodings for adjacent target label values should be regularized to find a balance between the complexity of the decision boundary and the error-correction capability of designed codes for a given benchmark.*

Regularization Function Derivation

The second property suggests to regularized L1 distance between encodings of adjacent label values. However, In a randomly formed minibatch consisting of only a subset of training examples, adjacent target labels might not be present. Hence it

is nontrivial to apply this regularizer to the label encoding.

The output encodings $\hat{Z}_i$are multiplied with D to generate the correlation vector $\hat{C}_i$ (Figure 4.8). We use the multiclass classification loss between $\hat{C}_i$ and the target labels for training. Due to this, label encoding E and decoding matrix D are related, and use of matrix D proves to be effective for regularizer R2. We further explain this in detail below.

Let E represent an encoding matrix of size $N \times M$. Each row $E_{k:}$ represents the encoding output (average) when the label is k. D is the decoding matrix of size $M \times N$. Let $\hat{C}_k$ represent the output correlation row vector of size $1 \times N$ when the target label is k. Here, $\hat{C}_k$ is obtained by multiplying $E_{k,:}$ with D (Figure 4.8).

$$\hat{C}_k = E_{k,:}D \tag{4.12}$$

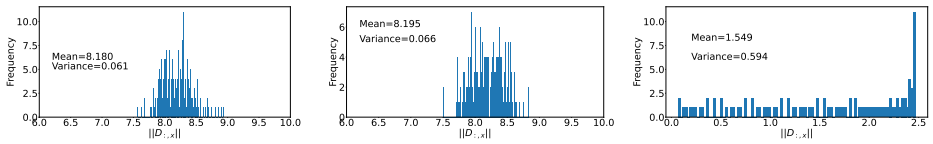
Since we apply softmax on the output vector to find the predicted label (Figure 4.8), ideally, $\hat{C}^k_k$ should have the highest value as the target label value is k.

$\therefore \hat{C}^k_k > \hat{C}^x_k,$ where, $x \neq k, x \in \{1,2,...,N\}$

$\therefore E_{k,:}.D_{:,k} > E_{k,:}.D_{:,x},$ where, $x \neq k, x \in \{1,2,...,N\}$ (Using equation 4.12).
Let $\theta_{k,x}$ represent the angle between row vector $E_{k,:}$ and column vector $D_{:,x}$. This leads to the below equation:

$$||E_{k,:}||||D_{:,k}||\cos(\theta_{k,k}) > ||E_{k,:}||||D_{:,x}||\cos(\theta_{k,x}), \text{ where, } x \neq k, x \in \{1,2,...,N\} \tag{4.13}$$

In our manually designed label encodings, the decoding matrix D is set to the encoding matrix (e.g., Figure 4.6). Here the decoding matrix is designed to have equal number of 1s and 0s in each column for binary-encoded labels. Hence the L2 norm of each column is the same. In label encoding learning, parameters of matrix D are learned during training and are not constrained to have the same L2 norm for each column. However, we observe a similar trend empirically. Figure 4.9 plots the distribution of $||D_{:,x}||$ for different benchmarks. As shown in the figure, for most benchmarks, we observe a small variance in the distribution of $||D_{:,x}||$. Based on this intuition and empirical validation, we assume that $||D_{:,x}|| \approx ||D_{:,y}||$ for $x \in [1,N]$



(a) LFH2 benchmark - Pitch label

(b) LFH2 benchmark - Roll label

(c) LFH1 benchmark - Roll label

Figure 4.9: (a) and (b) plot the distribution of $||D_{:,x}||$ for LFH2 benchmark. (c) plots the distribution of $||D_{:,x}||$ for LFH1 benchmark. Here the variance is very low, which suggests that the assumption $||D_{:,x}|| \approx ||D_{:y}||, x \in [1,N], y \in [1,N]$ is valid. For the LFH1 benchmark, the variance is higher than LFH2. However, all outliers are for label values with very few (or even zero) training examples.

and $y \in [1,N]$ to simplify the analysis. Using this assumption in equation 4.13 leads to the following inequality:

$$\cos(\theta_{k,k}) > \cos(\theta_{k,x}), \text{ where, } x \neq k, x \in \{1,2,...,N\}$$

Thus the cosine similarity between $E_{k,:}$ and $D_{:,k}$ should be the highest to predict the label k. The optimization process to reduce the loss between the target and prediction will try to maximize this cosine similarity. In the best case, the angle between $E_{k,:}$ and $D_{:,k}$ will be zero, and both vectors are parallel.

This simplification leads to the following relation between E and D.

$E_{k,:} = tD_{:,k}$, where $t > 0$

Similarly, $E_{k+1,:} = t'D_{:,k+1}$, where $t' > 0$

Since t and t' both are positive values, reducing $D_{i,k} - D_{i,k+1}$ also reduces $E_{k,i} - E_{k+1,i}$. Thus, the regularization rule outline by Equation equation 4.11 can be imposed using the following equation:

$$\mathscr{L}_2 = \sum_{i=1}^{M} \sum_{j=1}^{N-1} |D_{i,j} - D_{i,j+1}| \tag{4.14}$$

4.4.3 Loss Function Formulation

We use the cross-entropy loss between $\hat{C}_i$ and soft target labels. Here, each bit of label encoding resembles a binary classifier. However, identifying the predicted label corresponding to the multi-bit label encoding can be treated as a multiclass classification problem. Soft target labels are probability distributions generated using the distance between different classes. Soft target labels can be used with cross-entropy loss and have shown improvement over typical classification loss between the correlation vector $\hat{C}_i$ and quantized target label Q_i or regression loss between the expected prediction $\hat{y}_i$ and target label y_i for ordinal regression [56]. We use this loss function for RLEL and multiclass classification. Complete loss function with regularizers (Equation 4.10 and 4.14) can be written as:

$$\begin{aligned}\mathscr{L} = \sum_{i=1}^{K} \mathrm{CE}(\hat{C}_i, \phi(y_i)) + \alpha \sum_{i=1}^{M} \sum_{j=1}^{N-1} |D_{i,j} - D_{i,j+1}| \\ + \beta \sum_{i=1}^{K} \sum_{j=1}^{K} \max(0, 2 \times |y_i - y_j| - ||\hat{Z}_i - \hat{Z}_j||_1), \\ \text{where } \phi^j(y_i) = \frac{e^{-|j-y_i|}}{\sum_{n=1}^{N} e^{-|n-y_i|}}\end{aligned} \tag{4.15}$$

Here, the first term is the loss between target and predicted labels. ϕ_i represents the target probability distribution generated from target y_i. The second and third terms are for regularizer R1 (Equation 4.10) and regularizer R2 (Equation 4.14), respectively.

A trade-off exists between the proposed desirable properties of label encodings: Encouraging one design property comes at the cost of relaxing constraints imposed by other design properties. As demonstrated by [223], finding the right balance between these properties for a given benchmark is crucial to finding the best label encoding for a given problem. Thus, these design properties can be naturally applied as regularizers, and the search for balance between different properties can be seen

as tuning the regularization parameters α and β.

4.5 Experimental Setup

We evaluate the regression error achieved by different manually designed encoding functions, decoding functions, and learned label encoding with RLEL. We compare RLEL with other encoding design approaches, including simulated annealing and autoencoder. We also compare BEL (i.e., manually designed functions) RLEL with generic regression approaches, such as direct regression and multiclass classification. For direct regression, L1 or L2 loss functions with L2 regularization are used. Label value scaling (hyperparameter) is used to change the numeric range of labels. For multiclass classification, we use cross-entropy loss between the softmax output and target labels.

The feature extractor and regressor are trained end-to-end for all approaches. The feature extractor architecture, data augmentation, and the number of training iterations are kept uniform across different approaches for a given benchmark. There is no notable difference between the training time for all approaches. The training dataset is divided into 70% training and 30% validation sets for tuning hyperparameters. The network is trained using the full dataset after hyperparameter tuning. We use the same values for quantization levels as prior work [223]. An average of five training runs with an error margin of 95% confidence interval is reported.

4.5.1 Benchmarks

Table 4.2 summarizes the regression tasks, feature extractors architecture (Figure 4.8), and datasets for benchmarks used for evaluation. Selected benchmarks cover different tasks, datasets, and network architectures and have been used by prior works on regression due to the complexity of the task [56]. We also evaluated RLEL on facial landmark detection tasks with smaller datasets to demonstrate its generalization capability. In this setup, a subset of training samples is used for training, whereas the complete test dataset is used to measure the test error.

Table 4.2: Benchmarks used for evaluation. Here the suffix "_s" for FLD1-FLD3 represents training with 10% of the original training dataset.

Task	Feature Extractor	Dataset	Benchmark	Label range/ Quantization levels	θ
Landmark-free 2D head pose estimation	ResNet50 [103]	BIWI [64]	LFH1	0-200/200	10
		300LP/AFLW2000 [285]	LFH2	0-150/150	10
	RAFANet [14]	BIWI [64]	LFH3	-180-180/360	50
		300LP/AFLW2000 [285]	LFH4	-180-180/360	50
Facial Landmark Detection	HRNetV2-W18 [264]	COFW [29]	FLD1/FLD1_s	0-256/256	10
		300W [216]	FLD2/FLD2_s	0-256/256	10
		WFLW [271]	FLD3/FLD3_s	0-256/256	10
		AFLW [138]	FLD4	0-256/256	10
Age estimation	ResNet50/ ResNet34	MORPH-II [210]	AE1	0-64/64	10
		AFAD [177]	AE2	0-32/32	10
End-to-end learning for self-driving car steering	PilotNet[23]	PilotNet	PN	0-670/670	10

Headpose Estimation: In landmark-free 2D head pose estimation, for a given 2D image, the head pose of a human is directly estimated in terms of three angles: yaw, pitch, and roll. We use loose cropping around the center with random flipping for data augmentation. We use the ResNet50 and RAFA-Net networks as the feature extractor. For ResNet50, the network is initialized using pre-trained parameters for ImageNet [215] dataset. We use the evaluation methodology followed by prior works [213, 276]. Two protocols are used for evaluation. The first protocol (LFH1 and LFH3) uses the BIWI [64] dataset for training and evaluation. This dataset consists of $15,128$ frames of 20 subjects. Random $70\% - 30\%$ splits are used for training and evaluation. In the second protocol (LFH2 and LFH4), the network is trained using the 300W-LP [285] dataset consisting of $122,450$ samples. AFLW2000 [285] dataset is used for evaluation.

*Facial Landmark Detection (*FLD*):* Facial landmark detection focuses on finding the (x,y) coordinates of facial keypoints for a given 2D image. We use three datasets widely used for facial landmark detection: COFW [29], 300W [216], and WFLW [271]. HRNetV2-W18 network architecture for feature extraction [264] and the modified regressor architecture for label encoding proposed by BEL [223] are used in this work. Random flipping, scaling $(0.75 - 1.25)$, and rotation (± 30) are

used for data augmentation. The COFW dataset consists of $1,345$ training and 507 testing images annotated with 29 landmarks. The training set of the 300W dataset has $3,148$ images annotated with 68 facial landmarks. This dataset provides four test sets: full test set, common subset, challenging subset, and the official test set with 300 indoor and 300 outdoor images. WFLW dataset is a comparatively larger dataset with $7,500$ training and $2,500$ testing images. Each image is annotated with 98 facial landmarks. The test set is divided into six subsets: large pose, expression, illumination, make-up, occlusion, and blur.

Age Estimation AE*:* This task focuses on predicting a person's age from a given 2D image. MORPH-II [210] and AFAD [177] datasets are used for evaluation. We follow the protocols for preprocessing and data augmentation of datasets as per prior works [204, 223]. MORPH-II dataset consists of $55,608$ images with random split of $39,617$ training, $4,398$ validation, and $11,001$ test images. The AFAD dataset consists of $164,432$ images with random split of $118,492$ training, $13,166$ validation, and $32,763$ test images.

End-to-end Learning for Self-Driving Car Steering For the regression task of end-to-end learning for autonomous driving, we use the NVIDIA PilotNet dataset, and PilotNet (PN) model [22]. In this task, for a given image of the road, the angle of the steering wheel that should be taken next is predicted. The PilotNet driving dataset consists of $45,500$ images taken around Rancho Palos Verdes and San Pedro, California [40]. We use the data augmentation technique used by prior works [22].

4.5.2 Evaluation Metrics

We report the Mean Absolute Error (MAE) between the targets (y_i) and predictions ($\hat{y}_i$) for headpose estimation, age estimaton, and end-to-end learning for self-driving cars' steering. Let N represent the number of samples, and P represent the number of labels (three in head pose estimation). The MAE is defined as:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{P}\sum_{j=1}^{P}|y_{i,j} - \hat{y}_{i,j}| \tag{4.16}$$

We report the Normalized Mean Error (NME) between the targets y_i and predictions $\hat{y}_i$. Inter-ocular distance normalization is used for all datasets. For N test samples, P facial landmarks, and L normalization factor, the NME is defined as:

$$\text{NME} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{P}\cdot\frac{1}{L}\sum_{j=1}^{P}|y_{i,j}-\hat{y}_{i,j}|_2 \tag{4.17}$$

4.6 Evaluation

This section provides a detailed evaluation of the proposed hand-crafted label encodings, and compare it with existing regression approaches. Further, we evaluate the proposed approach RLEL for label encoding learning, and provide a detailed analysis of the regularization functions.

4.6.1 Impact of Encoding, Decoding, and Loss Functions on Regression Error

BEL introduces several design parameters for regression by binary classification. We evaluate different encoding ($\mathscr{E}$), decoding ($\mathscr{D}$), and training loss ($\mathscr{L}$) functions for BEL across all the benchmarks and study the extent and nature of the impact of these design parameters on accuracy.

Encoding function ($\mathscr{E}$): Figure 4.10 plots error (MAE or NME) using different encodings. We do not show results for Hadamard codes here as it results in significantly higher error than other encodings. Section A.2 provides tabular data of different encoding (including Hadamard code), decoding, and loss functions. On average, Hadamard codes result in $\sim 60\%$ higher error than J encoding, which shows that these codes are unsuitable for regression. The results show the encoding function significantly affects the accuracy and the best-performing encoding function varies across tasks, datasets, and network architectures (e.g., LFH1 and LFH3 are trained on the same dataset and different architecture). In Section 4.2 we observed that which encoding/decoding functions result in lower error depends upon the classifiers' error distribution. For decoding functions used for the comparison in Section 4.2, J does better than U for LFH3, FLD1, and AE1; we attribute this to

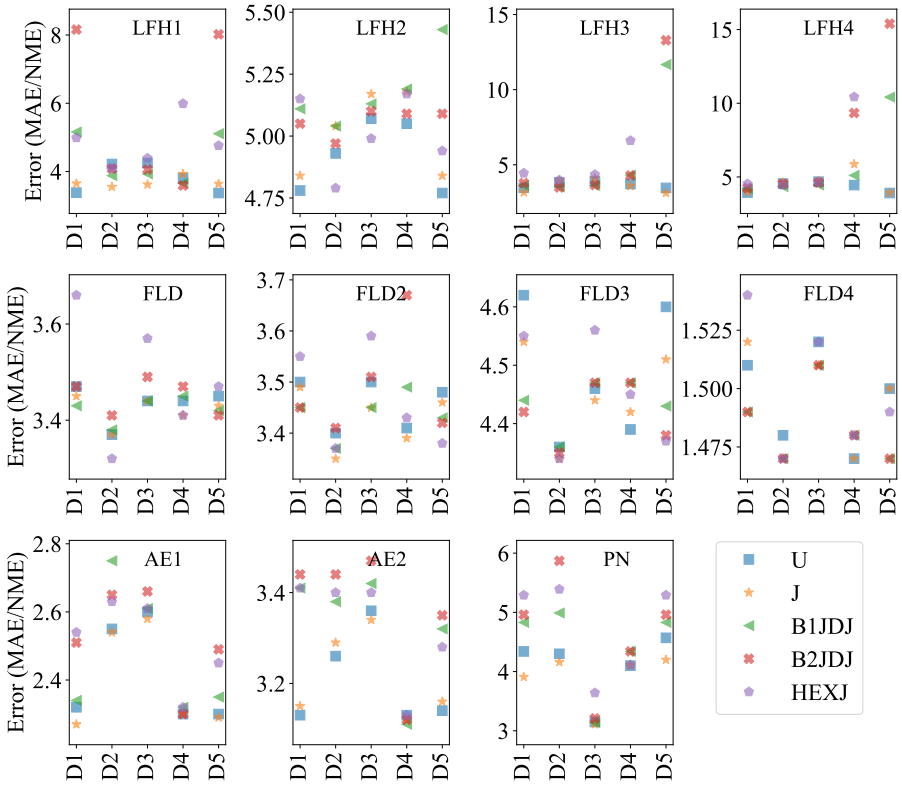


Figure 4.10: Error (MAE or NME) for different encoding, decoding, and loss functions for BEL. D1-D5 represents different combinations of decoding and loss functions: D1 (BCE loss with BEL-U/BEL-J/GEN decoding for U/J/others), D2 (CE/GEN-EX), D3 (CE/GEN), D4 (L1 or L2/GEN-EX), and D5 (BCE/GEN-EX).

misclassification errors occurring more frequently near bit transitions based on the analytical study.

The encoding function impacts the number of classifiers and the complexity of the function to be learned by a classifier. We observe a trade-off between these two parameters. For some benchmarks, the availability of sufficient training data and network capacity facilitates the learning of complex classifiers such as B2JDJ. In such a case, a reduced number of classifiers compared to U, J, or B1JDJ codes results in a lower error.

Decoding ($\mathscr{D}$) and training loss ($\mathscr{L}$) functions: We explore three decoding and three training loss functions (Section 4.3.2). However, not all the combinations of decoding and loss functions ($\mathscr{D}/\mathscr{L}$) perform well. For example, CE, L1, or L2 losses do not use decoding $\mathscr{D}^{\text{BEL-J}}$ or $\mathscr{D}^{\text{BEL-U}}$. Therefore, optimizing the network for these losses does not directly minimize the absolute error between targets and decoded predictions. We present results for five out of nine $\mathscr{D}/\mathscr{L}$ combinations. Figure 4.10 compares error (MAE or NME) achieved by different $\mathscr{D}/\mathscr{L}$ combinations and highlights the range of error variations. $\mathscr{D}^{\text{GEN-EX}}$ results in the lowest error for the majority of the benchmarks as it reduces quantization error and also utilizes the output logit confidence values. $\mathscr{D}^{\text{GEN-EX}}$ consistently perform better than $\mathscr{D}^{\text{GEN}}$ function that has been used for multiclass classification by prior works [4]. The use of CE or L1/L2 loss results in a lower error with $\mathscr{D}^{\text{GEN-EX}}$ for most benchmarks as the training loss function directly minimizes the error between targets and decoded predictions.

4.6.2 Comparison of Label Encodings

Table 4.3 compares different encoding design approaches. BEL-x represents an encoding function chosen from the proposed manually designed encoding functions (Section 4.3.1). Validation error is used to select the encoding function. RLEL results in lower error than simulated annealing and autoencoder-based approaches for most benchmarks. Both approaches are widely used for code design. However, for regression tasks, the suitability of label encoding depends upon the problem, including the task, network architecture, and dataset, as shown in the previous section. Simulated annealing or autoencoder-based approaches do not optimize the encodings end-to-end with the regression problem., resulting in higher error. Furthermore, the gap between the error of learned label encoding with and without regularizers (RLEL and LEL) increases for smaller datasets, which suggests that RLEL-learned codes generalize better.

RLEL can not be used with binary-cross entropy loss for training. We observe that for some benchmarks, the autoencoder-based approach outperforms (e.g., LFH1) as it can be used with binary-cross entropy loss. The main objective of RLEL is to automatically learn label encoding that can reach the accuracies of man-

Table 4.3: Comparison of RLELwith different label encoding design approaches. The bold and underlined numbers represent the first and second best errors, respectively.

	Error (MAE or NME)					
Approach	LFH1	LFH2	FLD1	FLD1_s	FLD2	FLD2_s
Sim. anneal.	4.32±0.12	5.03±0.08	3.55±0.01	6.52±0.05	3.59±0.00	5.35±0.01
Autoencoder	**3.38**±0.01	4.84±0.02	3.39±0.01	4.85±0.03	<u>3.39</u>±0.00	<u>4.20</u>±0.05
LEL	4.03±0.15	4.96±0.08	<u>3.36</u>±0.01	4.98±0.07	<u>3.39</u>±0.01	4.28±0.05
BEL-x	3.56±0.11	**4.77**±0.05	**3.34**±0.01	**4.63**±0.03	3.40±0.02	**4.15**±0.01
RLEL	<u>3.55</u>±0.10	**4.77**±0.05	<u>3.36</u>±0.01	<u>4.71</u>±0.04	**3.37**±0.02	**4.15**±0.05
Approach	FLD3	FLD3_s	AE1	AE2	PN	
Sim. anneal.	4.52±0.02	6.38±0.01	2.33±0.01	3.17±0.01	4.25±0.01	
Autoencoder	4.36±0.01	<u>5.62</u>±0.01	2.29±0.00	3.19±0.01	4.49±0.04	
LEL	**4.35**±0.02	5.68±0.04	2.30±0.01	3.17±0.01	3.22±0.02	
BEL-x	4.36±0.02	<u>5.62</u>±0.00	**2.27**±0.01	**3.11**±0.00	<u>3.11</u>±0.01	
RLEL	**4.35**±0.01	**5.58**±0.01	<u>2.28</u>±0.01	<u>3.14</u>±0.01	**3.01**±0.03	

ually designed codes (BEL), as using such codes is time and resource-consuming. Hyperparameter search for RLEL can be performed by off-the-shelf hyperparameter tuners/libraries without manual efforts [63, 150]. In contrast, hand-designed codes need human intervention to design codes. Also, multiple training runs are still required to find suitable codes for a given benchmark from a set of hand-designed codes. As shown in Table 4.3, *RLEL results in lower or comparable errors to hand-designed codes.*

4.6.3 Comparison with Regression Approaches

BEL and RLEL are generic regression approaches that focus on regression by binary classification. We compare BEL and RLEL with other generic regression approaches, including direct regression and multiclass classification as shown in Table 4.4. BEL-x results in 9.9% and 15.5% reduction in the regression error compared to direct regression and multiclass classification on average. RLEL consistently lowers the error compared to direct regression and multiclass classification with 10.9% and 12.4% improvement on average.

Table 4.4: Comparison of Manually-designed encodings and RLELwith different regression approaches and state-of-the-art task-specialized approaches. "/*x*M" represents the model size. BEL-x represents the encoding function with the least validation error chosen from the proposed manually designed encodings (Section 4.3.1).

	LFH1	LFH2	LFH3	LFH4
Direct Regression	4.22±0.13/23.5M	5.32±0.12/23.5M	3.40±0.26/69.8M	4.14±0.12/69.8M
Multiclass Class.	4.49±0.24/24.2M	5.31±0.05/24.8M	4.54±0.04/72.0M	5.14±0.08/72.0M
BEL-x	3.56±0.10/23.6M	4.77±0.05/23.6M	3.30±0.04/69.8M	3.90±0.03/69.8M
RLEL	3.55±0.10/23.6M	4.77±0.05/23.6M	-	-
Task-specialized**	3.30±0.04/69.8M	3.90±0.03/69.8M	-	-
	FLD1	FLD1_s	FLD2	FLD2_s
Direct Regression	3.60±0.02/10.2M	32.70±1.37/10.2M	3.54±0.03/10.2M	5.04±0.02/10.2M
Multiclass Class.	3.48±0.03/25.6M	5.36±0.03/25.6M	3.46±0.02/45.2M	4.50±0.04/45.2M
BEL-x	3.36±0.01/10.6M	4.71±0.04/10.6M	3.37±0.02/11.2M	4.15±0.05/11.2M
RLEL	3.36±0.01/10.6M	4.71±0.04/10.6M	3.37±0.02/11.2M	4.15±0.05/11.2M
Task-specialized**	3.45/9.6M	-	3.07/25.1M	-
	FLD3	FLD3_s	FLD4	AE1
Direct Regression	4.64±0.03/10.2M	6.35±0.07/10.2M	1.51±0.01/10.2M	2.37±0.01/23.5M
Multiclass Class.	4.46±0.01/61.3M	6.05±0.01/61.3M	1.56±0.01/20.1M	2.75±0.03/24.2M
BEL-x	4.35±0.01/11.7M	5.58±0.01/11.7M	1.47±0.00/10.8M	2.27±0.01/23.6M
RLEL	4.35±0.01/11.7M	5.58±0.01/11.7M	-	2.28±0.01/23.6M
Task-specialized**	4.32/-	-	1.57/9.6M	1.96/3.7M
	AE2	PN		
Direct Regression	3.16±0.02/23.5M	4.24±0.45/1.8M		
Multiclass Class.	3.38±0.05/24.8M	5.54±0.03/1.8M		
BEL-x	3.14±0.01/23.6M	3.01±0.03/1.9M		
RLEL	3.14±0.01/23.6M	3.01±0.03/1.9M		
Task-specialized**	3.47/21.3M	4.24/1.8M		

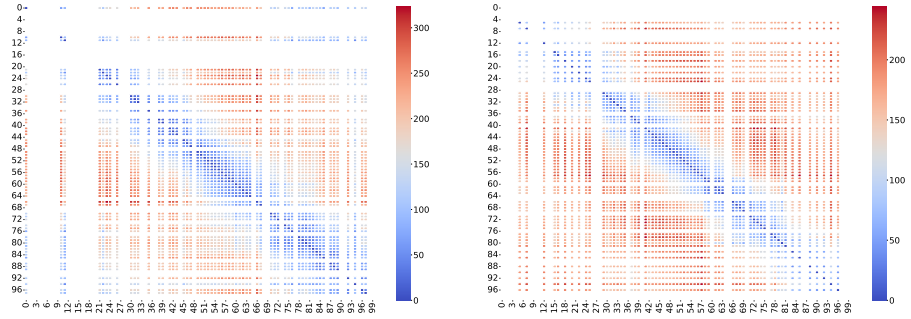
*This uses different network architecture, data augmentation, and training process.

4.6.4 Analysis of Proposed Regularizers

In this section, we further analyze the impact of proposed regularizers for RLEL on learned label encoding and regression error.

Impact of Regularizer R1

We proposed regularization function R1 to encourage the L1 distance between encodings to be proportional to the difference between corresponding label values. Figure 4.11a and Figure 4.11b represent the L1 distance between pairs of learned encodings for FLD1_s benchmark without and with regularization, respectively. The



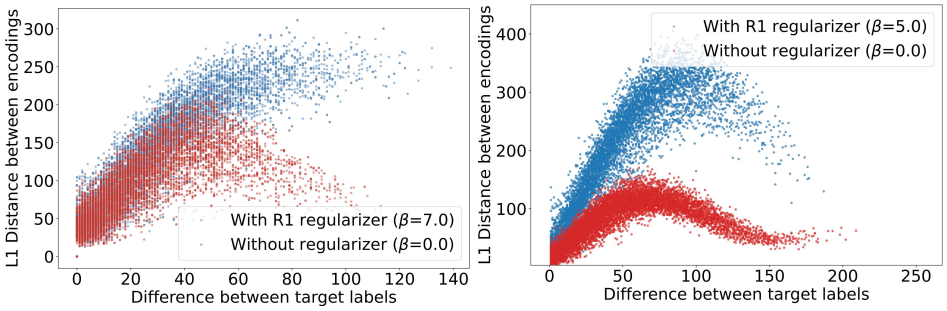
(a) Without regularization: $\beta = 0$ **(b)** With regularization: $\beta = 5.0$

Figure 4.11: (a) and (b) show the L1 distance between pairs of encodings for FLD1_s benchmark for $\beta = 0$ and $\beta = 5.0$, respectively. Each cell (i,j) in this matrix represents the L1 distance between learned encodings for label i and j, i.e., $||E_{i,:} - E_{j,:}||_1$.

X-axis and Y-axis represent the label values. Here, some columns and rows are replaced by white lines, as these label values are not present in the training dataset. The data point at coordinates (i, j) represent the L1 distance between encodings for label i and j, i.e., $||E_{i,:} - E_{j,:}||_1$. For example, in Figure 4.11a, the L1 distance between encodings for label values 0 and 97 is ~ 120 (light-blue coloured point at coordinate $(0, 97)$). In Figure 4.11b, the L1 distance between encodings for label values 4 and 96 is ~ 170 (red coloured point at coordinate $(4, 96)$).

The first design property (Section 4.4) states that the L1 distance between encodings should increase with the difference between corresponding label values. The difference between label values for pairs of encodings increases with the distance from the diagonal of this plot. Thus, the value of data points (i.e., the L1 distance between encodings) should increase with the distance from the diagonal of this plot. As shown in Figure 4.11a, without regularization, the distance between encodings is less for faraway label values (blue-colored data points away from diagonal), which shows that learned encodings do not follow the proposed design property. As shown in Figure 4.11b, the introduction of regularization function R2 remedies this and increases the L1 distance between encodings for faraway labels.

Figure 4.12 plots the L1 distance between encodings versus the difference between corresponding label values for benchmarks FLD1_s and FLD2_s. For both



(a) FLD_1 benchmark **(b)** FLD_1 benchmark

Figure 4.12: (a) and (b) plot the L1 distance between pairs of encodings versus distance between corresponding label values for FLD1_s and FLD2_s benchmarks.

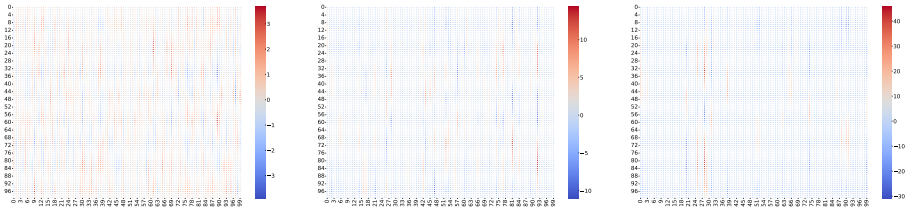
Table 4.5: Effect of regularization R2 on bit-transitions in binarized and real-valued label encodings.

α value	Proposed regularizer using Decoding matrix (Equation 4.14)	#Bit transitions in binarized label encoding	Approximated bit transitions in label encoding (Equation 4.11)
0	6816.1	5097	391.88
0.1	215.3	3596	168.52
0.5	130.8	3180	104.19

the benchmarks, the proposed regularizer R1 helps enforce the first design property for real-valued label encodings and results in better label encodings with lower error (Table 4.3).

Impact of Regularizer R2

The regularization function R2 is proposed to reduce the number of bit transitions in the learned label encoding. Table 4.5 summarizes the effect of α (i.e., the weight of R2) on the number of bit transitions in the decoding matrix and binarized/real-valued label encoding. The second column is the number of bit transitions in the decoding matrix (Equation 4.14), which is used as the regularization function. The third and fourth columns are the total number of bit transitions in binarized and real-valued encodings (Equation 4.11). The table shows that the proposed regularizer on the decoding matrix also encourages fewer bit transitions in the label encoding.



(a) Without regularization: $\alpha = 0$, MAE= 4.13

(b) $\alpha = 0.1$, MAE= 3.76

(c) $\alpha = 1.0$, MAE= 3.56

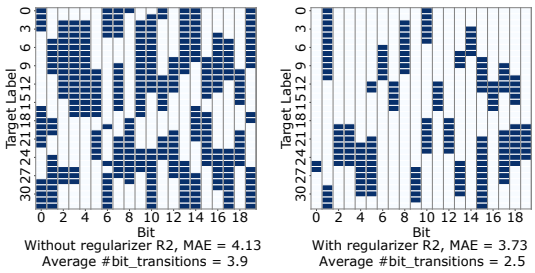
Figure 4.13: (a)-(d) represent the label encodings learned by RLEL for different values of weight α for regularizer R2 (Equation 4.15).

Figure 4.13 compares the label encodings learned for LFH1 benchmark for different values of α, where α is the weight of regularization function R2 (Equation 4.15). Each row k is the encoding for label value k. Each column k represents the output of the encoding position k for different label values. The regularization function is proposed to decrease the transitions in an encoding bit (blue→red and red→blue) over the range of label values. We observe the impact of proposed regularizer on learned label encodings shown in Figure 4.13; increasing the value of α decreases bit transitions in the learned label encodings and improves MAE. Figure 4.14a represents binarized representation of learned label encoding for clarity. The use of regularizer R2 reduces the number of bit-transitions (i.e., $1 \rightarrow 0$ and $0 \rightarrow 1$ transitions in a column) to enforce the second design property and consequently reduces the regression error.

There is a trade-off between the error probability and error correction capability of classifiers for regression. Hence, depending upon the benchmarks, more bit transitions can be added as the advantage of increased error correction outweighs the increase in classification error. We observe a similar trend, where adding R2 does not improve error for some benchmarks (FLD1_s, FLD2_s), as it constrains the number of bit transitions.

4.6.5 Ablation Study

This section provides different ablation studies for the proposed approaches. We show the impact of quantization levels, bottleneck layer size, dataset size on the



(a) Effect of regularizer R2

Figure 4.14: Regularizer R2 reduces the number of bit transitions per bit, reducing the complexity of decision boundaries to be learned by binary classifiers. Here blue and white colors represent 1 and 0, respectively.

manually designed and learned label encodings.

Impact of quantization and decoding functions in BEL:

As discussed in Section 4.1, a real-valued label is quantized to a discrete value in $\{1,2,...,N\}$ before applying the encoding function. In Table 4.6, we show the effect of increasing the number of quantization levels (N) on the error for correlation-based decoding ($\mathscr{D}^{\text{GEN}}$, which returns a quantized prediction) and expected correlation-based decoding ($\mathscr{D}^{\text{GEN-EX}}$, which returns a continuous prediction). As shown in the table, there exists a trade-off between reducing quantization error and using fewer classifiers. The error is lower for 128 quantization levels than it is for 256 as the improvement resulting from fewer binary classifiers is higher than the increase in quantization error. Moreover, the use of proposed decoding function $\mathscr{D}^{\text{GEN-EX}}$ for regression consistently results in lower error compared to $\mathscr{D}^{\text{GEN}}$.

Table 4.6: Impact of the quantization and decoding functions on NME for facial landmark detection.

	COFW			300W		
Quantization levels	64	128	256	64	128	256
$\mathscr{E}^{\text{BEL-U}} + \mathscr{D}^{\text{GEN}}$	3.66	3.51	3.46	3.79	3.59	3.46
$\mathscr{E}^{\text{BEL-U}} + \mathscr{D}^{\text{GEN-EX}}$	3.46	3.41	3.44	3.54	3.47	3.44
$\mathscr{E}^{\text{BEL-J}} + \mathscr{D}^{\text{GEN}}$	3.65	3.49	3.43	3.76	3.58	3.46
$\mathscr{E}^{\text{BEL-J}} + \mathscr{D}^{\text{GEN-EX}}$	3.45	**3.40**	3.42	3.52	3.45	**3.43**

Table 4.7: Impact of the number of quantization levels on error for FLD1 benchmark

Quantization levels (N)	NME
32	3.49
64	3.36
128	3.36
256	3.36
384	3.37
512	3.37

Impact of the number of quantization levels (N) in RLEL

We further analyze the effect of the number of quantization levels for RLEL. Table 4.7 shows the NME (Normalized Mean Error) for different values of quantization levels for FLD1 benchmark. This suggests that the proposed method RLEL is less sensitive to the number of quantization levels for higher values. For RLEL, the decoding matrix that converts the encodings to the predicted label is also learned during the training (Figure 3). This matrix is of size $M \times N$, where each column represents the weight parameters for one quantization level. One possible reason for the above results is that matrix D learns the number of quantization levels suitable for this problem.

There is a potential to learn the number of quantization levels and non-uniform quantization using the proposed RLEL framework. For example, in Figure 3-step (4), fixed parameters j are used to scale the correlation vector $\hat{C}_i^j$ and find the expected prediction $\hat{y}_i$. These parameters represent quantization levels. One possible approach to learning the quantization levels is to make these parameters trainable. In this case, L1/L2 loss between the expected prediction $\hat{y}_i$ and target labels y_i can be used to train the network.

Impact of dataset size on error for RLEL and BEL

In order to compare the effect of dataset size on encoding design, we run BEL and RLEL approaches with the same training loss function (cross entropy loss in equation 4.15). We take the dataset FLD1 and use a fraction of the dataset for training. The entire test dataset is used for testing here. Table 4.8 summarizes the

Table 4.8: Effect of dataset size on the error for FLD1 benchmark.

%Dataset used	RLEL	BEL	Difference (RLEL-BEL)
100	3.36	3.35	0.01
80	3.43	3.42	0.01
60	3.53	3.47	0.06
40	3.77	3.72	0.05
20	4.08	4.04	0.04
10	4.71	4.63	0.08

error achieved by RLEL and BEL for different fractions of the training dataset. The evaluation shows that the gap between the performance of RLEL and BEL decreases with the increase in dataset size, which suggests that RLEL might be able to achieve lower error for larger datasets.

4.7 Case Study: Model Compression using Label Encodings for Computer Vision Tasks

In this section, we evaluate the impact of label encodings on model compression for compute- and energy-efficient inference of a deep regression model. As discussed earlier, in regression using label encoding setup, predicting a set of values (e.g., classifiers' output) instead of one value (direct regression) introduces ensemble diversity, which improves accuracy [229]. Furthermore, encoded labels introduce redundancy in the label presentation, which improves error-correcting capability and accuracy [54]. This improvement in the accuracy and robustness can be exploited to further reduce the computations, and hence energy consumption, using widely used model compression approaches for inference [78, 94, 108].

4.7.1 Methodology

We use a subset of benchmarks used for evaluating BEL and RLEL. We use a one-shot pruning approach, where the network is pruned after training for certain epochs. The prune model is then fine-tuned. The total number of epochs used for training is the same as dense models. The first layer is not pruned. The last layer is not pruned in direct regression. However, the last layer consists of more parameters for BEL and RLEL in FLD benchmarks and is pruned for these benchmarks. Magnitude-based

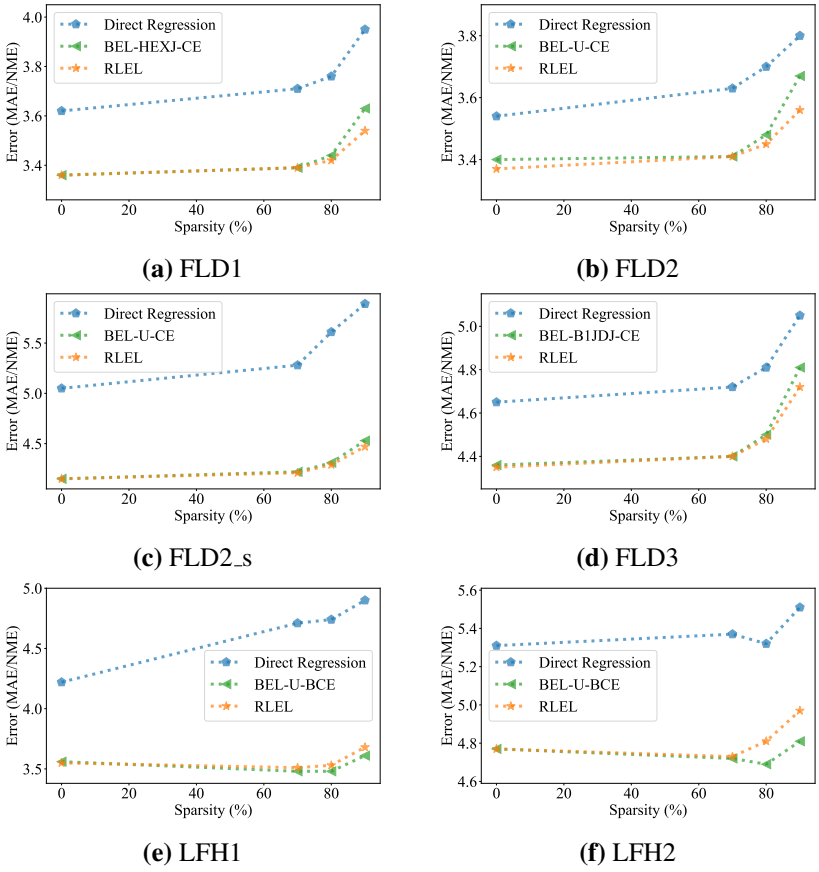


Figure 4.15: Regression error versus sparsity for direct regression, BEL (Manually designed encoding function), and RLEL (Learned Label encoding).

unstructured pruning is used for the rest of the layers.

4.7.2 Evaluation

Figure 4.15 compares the regression error for different sparsity levels for direct regression, manually designed label encodings, and learned label encodings.

For dense models, BEL and RLEL result in 10.1% 11.9% lower error compared to direct regression, respectively. Whereas, for 90% sparsity, BEL and RLEL result in 13.0% and 16.8% lower error compared to direct regression. For direct

regression, the regression error increases by 10.8% from 0% to 90% sparsity. For BEL, the regression error increases by 6.2% from 0% to 90% sparsity. On the other hand, for RLEL, the regression error increases by 5.8% from 0% to 90% sparsity. These results indicate that the use of label encodings reduces the degradation in accuracy for model compression approaches compared to direct regression, which indicates the robustness of the use of label encodings. Further, we observe that for benchmarks where the BCE loss function is used in the BEL setup (LFH1 and LFH2), the 90% sparse model has only 1.1% higher regression error compared to the dense model. Overall, 90% sparse model with RLEL has 5.1% lower regression error on average compared to dense models with direct regression. Thus, the use of label encodings enable compute- and energy-efficient regression networks. The next section provides a case study of using label encodings and model compression to reduce the computation in neural path planning.

4.8 Case Study: Label Encodings for Energy-Efficient Neural Path Planning

Several works have recently explored neural planners for informed sampling-based motion planning to improve the success rate achieved in fixed runtime budget [73, 198, 282]. These approaches combine traditional motion planning approaches with deep learning to improve motion planning quality while reducing the runtime and collision checks significantly compared to traditional motion planning approaches [198, 282]. However, these works do not focus on the hardware aspect of neural planners, and these neural planners are not optimized for compute- and energy-efficient execution. We profile 2D path planning using MPNet [198], a learning-based motion planning approach, on a CPU-GPU system, where collision detection takes 23$\times$ more time than neural planning. We further profile MPNet 2D path planning on an accelerator-rich system using microarchitectural simulation, where neural planning is executed on an EdgeTPU [83] and collision detection is executed on CECDU proposed in Section 3.3. On a CPU-EdgeTPU-CDU system, neural planning consumes $\sim$ 50% of the runtime and $\sim$ 33% of total power consumption. Thus, designing neural planners suitable for realtime energy-efficient robotic motion planning is important.

Typically, a regression network is used for neural planning, which predicts one or more continuous values (e.g., 2D coordinates of a milestone for 2D path planning). We propose a Binary-Encoded Label (BEL)-based neural planner, which uses a set of binary classifiers for neural planning. We evaluate the impact of the BEL neural planner on motion planning quality and computation. We show that the proposed BEL neural planner results in a more accurate neural planner, which improves the motion planning success rate and reduces the number of collision detection checks for 2D path planning problems. Further, our evaluation shows that the proposed neural planner is more robust to the reduction in the number of model parameters and sparsity, significantly reducing the computation requirement of the neural planner. BEL neural planner reduces the neural planning and collision detection computation by $11.8\times$ and 17.4% with equal or higher motion planning success rate. We further describe the methodology and evaluation of the BEL neural planner.

4.8.1 Methodology

This section describes the motion planning benchmarks and neural planner architecture used for evaluation.

Motion Planning Benchmarks: We consider a 2D path planning problem. The experimental methodology and benchmarks are the same as MPNet [198]. The neural planner is trained using 100 workspaces, where 4000 paths are generated using state-of-the-art classical motion planner RRT* [126] for different start and end positions in each workspace. For testing, we use two test datasets: 2D-seen and 2D-unseen. For the 2D-seen test set, 100 workspaces used for training are used for testing as well, where 200 unseen pairs of start and goal are used for each workspace. For the 2D-unseen test set, 10 new workspaces and 2000 pairs of start and end positions per workspace are used. Here, each workspace consists of 7 squares randomly placed in the environment. The extent of the environment is 40×40, and each obstacle is of size 5×5. Note that 2D-seen benchmarks might not be realistic for deployment. However, we compare peformance for 2D-seen and 2D-unseen benchmarks to demonstrate the generalization power of BEL neural planner.

Table 4.9: Neural planner architectures evaluated in this work. Param represents the number of weight parameters, and compute represents the number of FLOPs per inference. Model-Spx represents a sparse network with x% sparsity.

	Model	Name	Sparsity	Dropout	Param ($\times 10^3$)	Compute ($\times 10^3$FLOPs)
Baseline Neural Sampler	L	L-Dense	0%	50%	3754	1878
	L	L-Sp50	50%	50%	1877	939
	L	L-Sp80	80%	50%	751	376
	M	M-Dense	0%	50%	1550	776
	M	M-Sp50	50%	50%	775	388
	S	S-Dense	0%	50%	264	133
BEL Neural Sampler	L	L-Dense	0%	90%	3755	379
	L	L-Sp50	50%	90%	1878	189
	L	L-Sp80	80%	90%	751	76
	M	M-Dense	0%	90%	1552	158
	M	M-Sp50	50%	90%	776	79
	S	S-Dense	0%	80%	266	56

Neural Planner Architecture: The workspace obstacles information is encoded using an encoder network E_Net consisting of three layers. We use a pre-trained network for E_Net provided by MPNet [198]. We explore three different types of planning networks: Large (L), Medium (M), and Small (S) with 12, 9, and 6 fully connected layers, respectively. We also consider 50% and 80% sparsity for the planning network L and 50% sparsity for the planning network M. All neural planning networks are trained for 400 epochs. For pruning, all the layers of the neural planner, except the last one, are uniformly pruned (Weight magnitude-based pruning [94]) at epoch 100. The pruned network is trained for 300 epochs after this. Further, we also use Dropout in all layers (except the last one) to introduce stochasticity in the planning phase, as shown beneficial by prior works [198]. The size of the binary code is set to 40 for the BEL neural planner. Table 4.9 summarizes model size, dropout rate, and FLOPs per inference for different neural planner architectures explored in this work.

4.8.2 Evaluation

This section provides a detailed evaluation of BEL neural planner and its impact on motion planning quality, success rate, and computation. Further, we compare

computation energy for neural planning and collision detection for different neural planner models and hardware acceleration.

BEL Neural Planner

We propose to use Binary Encoded Labels for improved neural planning. We further exploit improved accuracy to reduce the network architecture parameters and computation using a smaller or sparser network.

Table 4.10 compares proposed BEL neural planner's motion planning quality and computation requirements with baseline (direct regression) for 2D-seen benchmarks. For the 2D-seen benchmark, the network is trained and tested for the same set of environmental scenarios (i.e., position of obstacles). We observe that the BEL neural planner slightly improves the motion planning success rate over the baseline for L-Dense. This improvement increases with the sparsity level (Dense to 80%). Further, the BEL neural planner significantly reduces the computation for motion planning by reducing the number of collision checks by 28% and the number of sampling inferences by 45% for L-Dense architecture. BEL neural planner results in a slight decrease in the motion planning success rate and 30.8% increase in the number of collision checks for S-Dense compared to L-Dense. However, S-Dense results in 3.8$\times$ reduction in the neural planner inference computation (Table 4.9). Note that since a path smoother is used after neural planning, the path quality, approximated using average milestones per path, does not change significantly across different neural planners.

Table 4.11 compares the motion planning quality and computation of different neural planners for 2D-unseen benchmarks. For 2D-unseen benchmarks, the environmental scenarios used for testing are not included in the training dataset. Thus, evaluation of these benchmarks signifies how well the trained neural planner generalizes. We observe a higher gap between the motion planning success rate achieved by baseline and BEL neural planners for 2D-unseen benchmarks compared to 2D-seen benchmarks. This shows that BEL neural planners generalize better for unknown environments outside of its training dataset. For L-Dense, the BEL neural planner provides a 3.2% higher success rate while providing 23% reduction in the collision checks and 52% reduction in the number of inferences per problem. We ob-

Table 4.10: Comparison of motion planning quality and computation for the baseline and BEL neural planners in 2D-seen benchmarks.

	Model	Motion Planning Quality		Motion Planning Computation	
		Success rate	Path-length (#Milestones)	#Collision checks	#Inference
Baseline Neural Sampler	L-Dense	99.18%	2.7	26339	8.4
	L-Sp50	98.19%	2.7	26138	10.8
	L-Sp80	98.27%	2.7	27035	11.1
	M-Dense	98.90%	2.7	28384	8.9
	M-Sp50	98.93%	2.7	29266	9.3
	S-Dense	97.74%	2.7	32568	11.0
BEL Neural Sampler	L-Dense	99.86%	2.7	18809	4.6
	L-Sp50	99.75%	2.7	20601	5.5
	L-Sp80	99.45%	2.7	21125	5.8
	M-Dense	99.45%	2.7	21567	6.2
	M-Sp50	99.60%	2.7	21713	6.3
	S-Dense	98.60%	2.7	24615	8.0

serve that for baseline neural planners using model L, the success rate increases for higher sparsity, possibly due to the regularization effect introduced by pruning [13]. For M and S models, the BEL neural planner provides $\sim 6\%$ higher success rate compared to baseline with $\sim 20\%$ reduction in the number of collision checks and $\sim 33\%$ reduction in the number of neural planner inferences. BEL neural planner maintains the motion planning success rate for L-Dense to M-Sp50 models ($4.5\times$ reduction in inference computation), which suggests that BEL neural planner can provide similar motion planning quality with much less computation requirements.

Stochasticity with Dropout in BEL Neural Planner

Prior works on learning-based motion planning [198] proposed to use Dropout [237] to introduce stochasticity in the sampled path. During inference, each node in the hidden layer of the neural planner is dropped with probability `dp`, determined by the dropout rate. Thus, the neural planner can explore multiple paths between two points until a feasible solution is found. Dropout also helps reduce the computation required during neural planning due to introduced sparsity [230]. A higher dropout rate introduces more dynamic sparsity and reduces computation. Note that this

Table 4.11: Comparison of motion planning quality and computation for the baseline and BEL neural planners in 2D-unseen benchmarks.

	Model	Motion Planning Quality		Motion Planning Computation	
		Success rate	Path-length (#Milestones)	#Collision checks	#Inference
Baseline Neural Sampler	L-Dense	96.62%	2.6	22423	8.6
	L-Sp50	97.74%	2.7	23947	9.7
	L-Sp80	97.62%	2.7	24462	9.7
	M-Dense	92.46%	2.6	25944	8.1
	M-Sp50	92.29%	2.6	24633	7.8
	S-Dense	90.91%	2.6	25802	8.7
BEL Neural Sampler	L-Dense	99.68%	2.6	17170	4.1
	L-Sp50	99.56%	2.7	18975	5.2
	L-Sp80	99.20%	2.6	19308	5.3
	M-Dense	98.85%	2.6	20149	5.9
	M-Sp50	98.74%	2.6	19460	5.6
	S-Dense	96.78%	2.6	21488	6.6

dropout-induced dynamic sparsity differs from the static sparsity introduced by pruning the network during training.

Table 4.12 compares the motion planning quality and computation achieved by baseline and BEL neural planners for different dropout rates. For a pure regression-based neural planner, dropout rate of 0.5 provides a balance between accuracy and stochasticity, resulting in the highest motion planning success rate. BEL neural planner predicts a binary code instead of a single real-valued, which makes the BEL neural planner more robust to the dynamic sparsity introduced by dropout. We find that for BEL planners, a higher dropout rate is required to introduce required stochasticity during planning. BEL neural planner results in the highest success rate for the dropout rate of 0.9, further reducing the computation required during inference.

Hardware-aware Neural Planner Selection

We observe that BEL neural planner with different models provides motion planning success rate in $\pm 1\%$ range and similar path quality (Table 4.11). Intuitively, the smallest neural planner should be used for hardware acceleration. However, these

Table 4.12: Effect of the dropout rate on motion planning quality and computation for the baseline and BEL neural planners (L-Dense model and 2D-seen benchmarks).

	Dropout Rate	Motion Planning Quality		Motion Planning Computation	
		Success rate	Path-length (#Milestones)	#Collision checks	#Inference
Baseline Neural Sampler	0.5	99.18%	2.7	26338	8.4
	0.7	98.14%	2.7	21183	6.5
	0.8	93.64%	2.7	18213	5.7
	0.9	79.33%	2.6	15302	5.3
BEL Neural Sampler	0.5	99.00%	2.7	26687	7.8
	0.7	98.80%	2.7	22080	6.1
	0.8	99.31%	2.7	19587	5.3
	0.9	99.86%	2.7	18809	4.6

models provide a trade-off between the computation cost of collision detection and neural planning. Fig. 4.16 compares the collision detection and neural planning computation requirements of different neural planner architectures for 2D-seen and 2D-unseen benchmarks. A larger and more accurate neural planner reduces the number of collision detection runs required for motion planning at the expense of the higher cost of neural planning (data points in bottom right). Similarly, a smaller neural planner saves the computation required for neural planning at the expense of more exploration and higher costs for collision detection (data points in top left). We further show how neural planner selection varies depending on the underlying hardware accelerators and their energy consumption.

Collision detection is used to find if the robot collides with environmental obstacles for a given position. Different factors that impact the energy consumption of a single robot-environment collision detection check include (not limited to) environment clutterness, environment and robot's geometric representation and precision, and granularity of collision detection for a path. For example, increasing the number of environmental obstacles by $2\times$ increases the collision detection cost by $\sim 2\times$ (Section 3.7.2). The use of a sphere instead of an oriented bounding box to represent the space occupied by the robot can reduce the cost of the intersection test by up to $27\times$ [58]. Similarly, increasing the step size of collision detection for a path

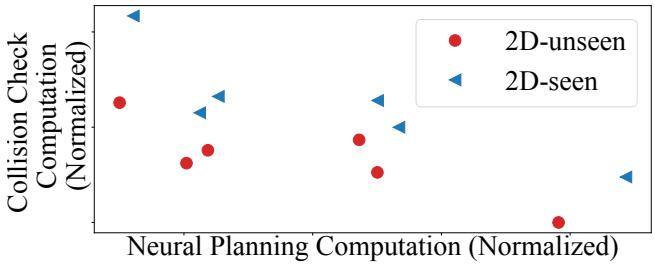
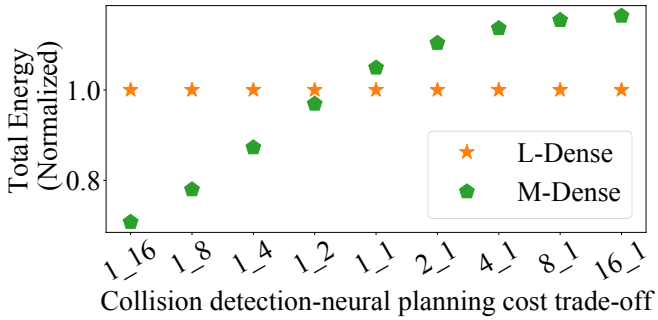


Figure 4.16: Comparison of Collision detection computation versus Neural planning computation for different neural planners explored in this work (Table 4.9).

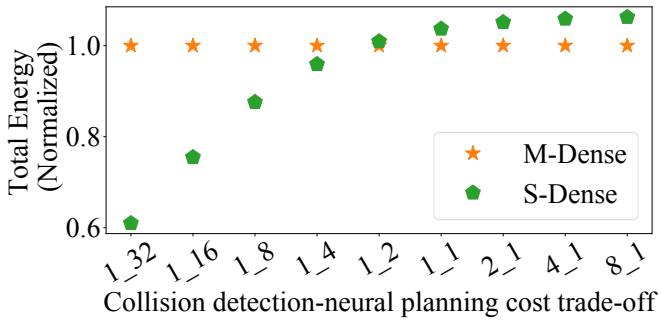
by $N\times$ also results in $\sim N\times$ reduction in the collision detection executions. Furthermore, the cost of collision detection varies across different hardware platforms. For example, oriented bounding box-environment octree collision check consumes $850\times$ more energy on a Cortex A57 core compared to a specialized hardware unit (Section 3.7).

Neural planning includes several inference executions of the neural planner. The energy consumption for neural planning inference depends upon several factors, such as numeric representation (e.g., FP32, FP16, INT32), and support for sparsity. For NVIDIA Ampere GPU, the TOPS/W increases by $2\times$ when moving from FP16 representation to Tensor Float 32 [178]. Several hardware accelerators have no or limited support for sparse inference. For example, NVIDIA Ampere Architecture supports up to 50% sparsity [178]. Different hardware accelerators provide varying performance in terms of TOPS/W. For example, TOPS/W for EdgeTPU is $\sim 4\times$ higher than TPUv3 [82, 83]. NVIDIA Jetson modules also have varying ranges of TOPS/W performance [179].

We approximate the total energy consumption of collision detection and neural planning for different neural planners. For the baseline configuration of the collision detection unit and neural planning hardware, we consider CECDU (Section 3.3) and neural network accelerator EdgeTPU [83]. For CECDU, we use a microarchitectural simulator estimate the energy consumption per OBB-environment collision detection. We use a step size of 0.05 for collision detection of a path connecting two



(a)



(b)

Figure 4.17: Comparison of total computation energy (Collision detection + Neural planning) for two different neural planners. For each configuration, the total energy is normalized to the energy consumption of L-Dense (a) and M-Dense (b). x_y represents a configuration where the ratio between collision detection and neural planner inference energy consumption. Configuration 1_1 represents the baseline configuration for collision detection and neural planning energy consumption.

milestones. For neural planning, we consider 1TOPS/W performance of the neural network accelerator [83]. We assume ideal support for sparsity and dropout for the baseline configuration (linear decrease in computation with sparsity/dropout rate). We further evaluate different configurations. Here, configuration x_y represents a setup where collision detection cost is $x\times$ more than the baseline configuration, and the neural planner inference cost is $y\times$ more than the baseline configuration. The baseline configuration is represented as 1_1.

Fig. 4.17 represents the approximated total energy consumption of collision de-

tection and neural planning of different neural planners for different configurations. Here, 2D-unseen benchmarks are used. For each configuration, the total energy is normalized to that of L-Dense (Fig. 4.17a) and M-Dense (Fig. 4.17b). As shown in both figures, given two neural planners, deciding which neural planner results in the lowest energy consumption depends upon the underlying cost collision detection and neural planner inference. For some configurations, the gap between the energy consumption of two neural planners can be as high as 30%, emphasizing the need for full system evaluation for neural planner selection.

4.8.3 Discussion

This section explores the use of label encoding for compute- and energy-efficient neural planners for 2D path planning. The proposed BEL neural planner improves the motion planning success rate by up to 6%. Further, for the same motion planning quality, the BEL neural planner results in a $4.5\times$ reduction in the computation for neural planning.

4.9 Related Work

Binary classification for regression: Prior works have proposed binary classification-based approaches for ordinal regression [44, 47, 149]. Ordinal regression is a class of supervised learning problems, where the samples are labeled by a rank that belongs to an ordinal scale. Ordinal regression approaches can be applied to regression by discretizing the numeric range of the real-valued labels [16, 72]. In the existing works on ordinal regression by binary classification, $N-1$ binary classifiers are used for target labels $\in \{1,2,...,N\}$, where classifier-k predicts if the label is greater than k or not for a given input. [149] provided a reduction framework and generalization bound for the same. However, the proposed binary classification formulation is restricted. It requires several binary classifiers if the numeric range of output is extensive, whereas reducing the number of classifiers by using fewer quantization levels increases quantization error. Thus, a more generalized approach for using binary classification for regression is desirable to allow flexibility in the design of classifiers.

Binary classification for multiclass classification: [54] proposed the use of

error-correcting output codes (ECOC) to convert a multiclass classification to a set of binary classification problems. This improves accuracy as it introduces tolerance to binary classifiers' errors depending upon the hamming distance (i.e., number of bits changed between two binary strings) between two codes. [4] provided a unifying framework and multiclass loss bounds in terms of binary classification loss. More recent works have also used Hadamard code, a widely used error-correcting code [229, 259]. Other works have focused on the use and design of compact codes that exhibit a sub-linear increase in the length of codes with the number of classes for extreme classification problems with a large number of classes [45, 61]. However, the proposed encoding and decoding approaches do not consider the task objective and labels' ordinality for regression. Further, the binary classifiers possess distinct error probability distribution properties for regression problems as observed empirically (Section 4.2), which can be exploited to design codes suitable for regression.

Multiclass classification and ordinal regression by binary classification can be viewed as special cases falling under the BEL framework. As shown in Section 4.6.1, other BEL designs yield improvements in accuracy over these approaches.

Encoding design: Encoding design is a well-studied problem with applications in several fields. Iterative approaches, such as simulated annealing or random walk, have been proposed for code design [54, 229]. However, iterative approaches are computationally expensive as each iteration requires full/partial training of the network to measure the error for sample encodings. Cissé et al. [45] proposed an autoencoder-based approach to design compact codes for multiclass classification problems with a large number of classes. However, these approaches do not consider the task objective and classifiers' nonuniform error probability distribution for regression. Deep hashing approaches aim to find binary hashing codes for given inputs such that the hashing codes preserve the similarities in the inputs space [123, 156, 158, 263, 272]. Deep supervised hashing approaches use the label information to design the loss function. In deep hashing, loss functions are designed to decrease the hamming distance between binary codes for similar images (e.g., same label). In contrast, RLEL is designed to reduce the error between decoded output codes and target labels.

We further summarize related work on task-specific approaches for regression

problems used in this work. While effective, task-specific approaches lack generality by design.

Head pose estimation: Head pose estimation is a widely studied problem. Existing task-specialized approaches propose different loss formulations or feature extractors to improve the error. HopeNet [213] proposed a combination of regression and classification loss. SSR-Net [275] and FSA-Net [276] proposed stage-wise soft regression. QuatNet [110] proposed to use MSE loss with custom ordinal regression loss. RAFA-Net [14] proposed an attention-based feature extractor architecture.

Facial landmark detection: Facial landmark detection is a widely studied problem. A common approach is to use heatmap regression, where the target heatmaps are generated by assuming a Gaussian distribution around the ground truth landmark location. Prior works proposed the use of binary heatmaps with pixel-wise binary cross-entropy loss [28]. HRNet [264] proposed a feature extractor that maintains high-resolution representations and uses heatmap regression. AWing [266] proposed a modified heatmap regression loss function with adapted wing loss. AnchorFace [274] used anchoring of facial landmarks on templates. LUVLi [136] proposed a landmark's location, uncertainty, and visibility likelihood-based loss.

Age estimation: Different approaches including ordinal regression [31, 75, 177, 185], soft stage-wise regression [275, 276], soft labels [56] have been proposed for age estimation. OR-CNN [177] and CORAL-CNN [31] proposed ordinal regression by binary classification with threshold-based encodings (i.e., unary codes). MV-Loss [185] proposed to penalize the model output based on the age distribution's variance. DLDL [75] augmented the loss function with KL-divergence between softmax output and soft target probability distributions. MV-Loss [185] proposed to penalize the prediction based on the variance of the age distribution.

4.10 Conclusion and Future Work

This work proposes binary-encoded labels (BEL) to pose regression as binary classification. We propose a taxonomy identifying the key design aspects for regression by binary classification and demonstrate the impact of classification error and encoding/decoding functions on the expected label error. Different encoding, decoding, and loss functions are explored to evaluate our approach using four

complex regression tasks. BEL results in an average 9.9%, 15.5%, and 7.2% lower error than direct regression, multiclass classification, and task-specific regression approaches, respectively. Our evaluation of several manually designed functions suggests that the best performing encoding/decoding function pair may be task, dataset, and network specific.

This work further proposes an end-to-end approach, Regularized Label Encoding Learning, to learn label encodings for regression by binary classification setup. We propose a combination of continuous approximation of binarized label encodings and regularization functions. This combination enables an efficient and automated search of suitable label encoding for a given benchmark using traditional continuous optimization approaches. The proposed regularization functions encourage label encoding learning with properties suitable for regression, and the learned label encodings generalize better, specifically for smaller datasets. RLEL-designed codes show lower or comparable errors to hand-designed codes. RLEL reduces error on average by 10.9% and 12.4% over direct regression and multiclass classification. BEL and RLEL improve accuracy over state-of-the-art approaches for head pose estimation (BIWI, AFLW2000), facial landmark detection (COFW), age estimation (AFAD), and end-to-end self-driving car steering (PilotNet).

We further evaluate the impact of model compression approaches on BEL and RLEL for a set of computer vision tasks and neural path planning. For computer vision benchmarks, BEL and RLEL result in lower degradation in the regression error with sparsity compared to direct regression, demonstrating its potential for energy-efficient inference acceleration. Further, for the same motion planning quality, the BEL neural planner results in a 4.5$\times$ reduction in the computation for neural planning.

Our analysis and empirical evaluation in this work demonstrate the potential of the vast design space of BEL for regression problems and the importance of finding suitable design parameters for a given task.

4.10.1 Future Research Directions

In this work, we introduce the design space of regression by binary classification and several design aspects that affect regression accuracy. This work mainly studies

encoding function design and its properties. Other design aspects introduced in this work can be explored further. For example, building better decoding functions that exploit error correction for regression problems is an interesting future direction. This work explored different loss functions and showed that the choice of the loss function is also benchmark-dependent. It is an interesting future direction to find other possible loss functions and automatically explore the design space of loss functions for a given problem setup.

The use of encoding introduces the quantization of real-valued labels. In the first part of the chapter, we focused on uniform quantization. In the second part of this chapter, automatic encoding learning is studied, which supports nonuniform quantization to some extent by learning sparse encoding. Some prior works on ordinal regression for monocular depth estimation [**?**] have demonstrated the use of logarithmic quantization. However, a systematic study of nonuniform quantization for different regression tasks in the proposed setup presents a possible future direction.

The proposed approach modifies the final layer's (i.e., regressor) architecture. In this work, we explored the use of a bottleneck layer to reduce the number of parameters in the regressor. We also observed that unlike conventional network architectures, where pruning the last layer reduces the accuracy significantly, BEL allows pruning of the regressor layers without reducing regression accuracy (improving accuracy in some cases). The design of the regressor (e.g., bottleneck layer) and its effect on the regression error for different problem setups present possible avenues for further improvement.

Several regression tasks have a very high number of labels. For example, many computer vision tasks perform per-pixel regression. BEL introduces network parameters overhead per regression label, and its integration for such regression tasks with a large number of labels is an exciting direction. Further, adapting the insights made for BEL for regression by binary classification in application domains such as neural rendering and reinforcement learning presents possible future research work.

The main goal of this work was to establish the design space of regression by binary classification and demonstrate the potential of systematic study of this design space. We have focused on using analytical-empirical studies to find out

the properties of several design aspects for regression; there is vast potential in the theoretical analysis of these design aspects. It would be interesting to gain a theoretical understanding of why a set of binary classifiers performs much better than regression, even though it introduces more things to learn. We believe that more improvements can be made by studying each of the design aspects and their relation with other design aspects and problem setup characteristics.

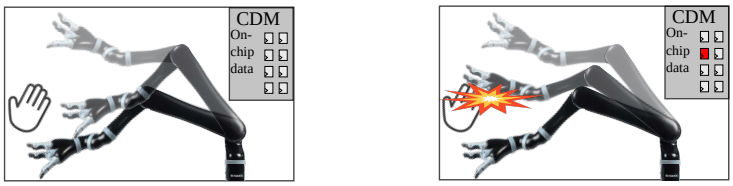
Chapter 5

Improving Soft-Error Resilience of Motion Planning Accelerators

Autonomous robots are increasingly used for real-time and safety-critical tasks, including medical care [55, 88], autonomous driving [246, 251], and home assistance [129, 267]. As autonomous robots are becoming an integral part of our lives, it is crucial to ensure that an autonomous robot does not collide with other objects, and thereby harm the safety of its surroundings.

Motion planning allows an autonomous robot to navigate and reach its end goal safely without collisions. Therefore, motion planning is key to the many tasks performed by autonomous robots, including navigation, object manipulation, footstep planning, and full-body movement. It has been an area of study since the 1970s [128, 141], and is today one of the key research topics in robotics.

The computational and real-time demands of motion planning exceed those provided by typical CPUs. Recently, several approaches have been proposed to accelerate motion planning on different hardware platforms, including GPUs [18, 76], FPGAs [8, 171, 226], and ASICs [11, 153, 173, 225, 233, 278]. Motion Planning Accelerators (MPAs) have achieved impressive performance gains and are being adopted in industry [59, 206, 212]. However, the use of MPAs in robotics applications has significant safety implications. For example, the IEC 61508 [115] provides functional safety standards for electronic systems used in applications such as robotics in terms of allowable dangerous failure rate per hour (Section 2.3.2).



(a) Error-free scenario **(b)** Faulty scenario

Figure 5.1: The effect of soft errors on safety in autonomous robots.

In modern systems, errors induced by particle strikes and radiation, or soft errors, make up the majority of SRAM and register-level faults [164, 236]. Hence, soft errors are a major threat to safety standards compliance in MPAs used in robotics. Furthermore, due to their transient nature, soft errors cannot be eliminated during the design and test phases of a chip, and hence need runtime mitigation.

There have been many studies on the effects of soft errors on CPUs, GPUs, and FPGAs [65, 67, 81, 96, 97, 125, 146, 148, 170, 182, 189, 190, 193, 218, 248, 249, 252, 257], deep learning accelerators [104, 105, 147, 154, 205, 214, 250, 260], and autonomous vehicle systems [12, 109, 119–121, 262]. However, their impact on accelerators for robotics has not been studied. Application-agnostic blanket error mitigation techniques such as error-correcting codes (ECC) and dual/triple modular redundancy (DMR/TMR) can increase the area, cost, and power by 12%-125% (Table 5.1), and degrade the performance of these hardware accelerators. With consumer applications driving growth of robotics, the electronics controlling these systems will become increasingly cost-sensitive [48, 80, 181]. Furthermore, any increase in the MPA's power consumption significantly reduces a mobile robot's operation time as MPAs suitable for real-time motion planning can contribute to 15%-50% of its total power consumption [132, 159, 171]. While there has been work on sensor and actuator faults in robotics [43], there has been no study of the reliability of MPAs in the presence of soft errors.

In this work, we focus on motion planning accelerators for motion planning approaches based on Probabilities Roadmaps (Section 2.1) We study soft errors in the collision detection module (CDM), which is the largest, most energy-consuming, and safety-critical component in MPAs [8, 153, 173, 233]. We find that CDMs in these accelerators consist of on-chip storage elements to store the information about

space that the robot passes through for possible motions. These storage elements account for more than 97% of the sequential elements (based on our RTL synthesis experiments), and so we focus on it in this paper. Figure 5.1 shows the effect of soft errors in the CDM. In the error-free scenario (Figure 5.1a), the robot navigates safely to the end goal. However, in Figure 5.1b, a soft error causes the MPA to misidentify a path taken by the robot as safe, thereby resulting in a collision. This mis-identification is due to an error modifying the geometric representation of the space that the robot passes through.

Prior work has explored memory and register file designs that allow flexible partition into protected and non-protected regions for incorporating selective error mitigation in systems using CPUs and GPUs [32, 168, 258, 280, 284]. These techniques protect only the most vulnerable data and do so by placing it in protected memory. A challenge with applying such an approach to error mitigation is that it requires accurate and fast identification of the most critical data. One approach is to use exhaustive fault injection (FI) to identify storage bits that exhibit the highest resilience improvement when protected from soft errors. Unfortunately, exhaustive FI can take up to $24,000$ CPU hours for a typical MPA (Section 5.5.1). Such high FI time overhead for error mitigation each time the MPA is reconfigured for a different task or robot introduces practical deployment challenges. For fully autonomous robots, the MPA's storage data can be generated or modified at runtime [69, 102, 118], requiring runtime characterization of vulnerable data for selective error mitigation. Also, as noted in earlier work [170] exhaustive FI is an inefficient way to gain insight during architecture design.

Prior work on FI techniques for CPUs and GPUs [65, 67, 97, 148, 176, 249, 257] has obtained significant reductions in the FI time. These FI tools and techniques are targeted towards specific languages, Instruction Set Architectures (ISA), or CPU/GPU system simulators and often exploit the microarchitecture or ISA to reduce the FI time and estimate the failure probability (Section 5.7). However, these techniques cannot be directly applied to robotic accelerators that use specialized microarchitectures and instruction sets. Thus, there is a need for techniques that efficiently characterize the effect of soft errors in robotics applications. Architectural Vulnerability Factor (AVF) has been widely used to define the vulnerability of a structure and can be measured using an analytical method such as Architec-

turally Correct Execution (ACE) analysis [170] or FI [145]. Directly applying AVF methodology such as ACE analysis to MPAs requires considering the positions of obstacles in the environment, thus leading to the need to run a large number of simulations to accurately estimate the fraction of time a hardware storage element contains an ACE bit.

To overcome these challenges, we introduce a novel heuristic, *Collision Exposure Factor (CEF)* that depends only upon the accelerator and robot, not on the environment. Other heuristics, such as bit position [147] and access-frequency [131, 163] have been proposed to find critical bits for deep learning accelerators and embedded applications, respectively. However, our analysis shows that considerable variation exists in the failure probabilities of bits with the same access-frequency or bit position in MPAs. In contrast, our approach provides a higher reduction in the overall failure probability as our proposed heuristic is more accurate at finding critical bits in the MPAs (Section 5.5.2).

The CEF estimates the vulnerability factor of memory bits storing spatial information. The 3D model data of the swept spaces of a robot's possible motions play a key role in real-time collision detection and motion planning, and is stored in the on-chip memory. Each bit in the on-chip storage helps specify the bounds of some motion's swept space. We define a bit's *critical space* as the region excluded from the swept space if that bit changes value due to a fault. A bit-error can lead to erroneous collision detection and safety violations if an obstacle overlaps with this bit's critical space. To account for this violation, CEF of a bit is defined as the surface area of that bit's critical space exposed to obstacles due to a soft error. The CEF is defined for a memory bit, and the probability of a fault in a bit resulting in a safety violation monotonically increases with its CEF value. Thus CEF values of memory bits can be used to estimate their relative vulnerability factors. In contrast, AVF estimates the failure probability due to soft error for a structure, and not in individual bits.

The probability of a fault in a bit resulting in a safety violation depends upon the environmental characteristics (e.g., average size and number of obstacles). By considering the entire swept space of the robot's motion, the CEF decouples the effects of the robot model and the environment on safety violations, and the safety violation probability of a bit monotonically increases with its CEF value. CEF

can be calculated once for a given MPA and robot *without* needing to consider obstacles in the environment (which might be highly dynamic). The underlying reason this separation is possible is that, in the most widely used approaches for real-time motion planning, a robot has a fixed set of possible short motions that are precomputed independently of obstacle positions [144]. During operation a subset of these motions is selected to navigate to a given goal depending upon obstacle positions. For purposes of fault analysis we decouple these steps by assuming a conservative distribution (e.g., uniform) on where objects will appear. We show empirically that the CEF computed this way *independent of the environment* is positively correlated with the average failure probability.

Further, we propose a CEF-aware error mitigation technique to selectively protect values with higher CEF in an MPA's on-chip storage. Finally, we propose a two-phase FI methodology: Phase 1 FI to find the CEF of all the bits (for a given robot and MPA), and Phase 2 FI to find the relation between the CEF and failure probability with fault site pruning. Uniform statistical FI-based characterization of the CDM provides similar speedup over exhaustive FI as CEF-aware FI (Section 5.5.1). However, it does not find the safety-violation probability of an individual bit nor does it find relative vulnerability of different bits, which is needed for error mitigation. In CEF-aware FI, on the other hand, decoupling the two FI phases allows efficient calculation of the safety-violation probability for every bit in the CDM. In summary, we make the following contributions in this paper:

- Establish the necessary conditions for safety violations (i.e., collisions), and propose Collision Exposure Factor (CEF), a reliability metric for CDM storage elements.
- Propose an efficient *CEF-aware error mitigation* technique that selectively protects values with higher CEF, and compare it to uniform, exhaustive FI, bit position-aware, and access-frequency-aware application of DMR, TMR, and ECC techniques for four CDM designs.
- Propose a two-phase FI methodology using the CEF of storage elements for fault site pruning to reduce FI time by orders of magnitude.

Our results show that CEF-aware selective error mitigation results in 12.3×,

9.6×, and 4.2× lower failure rate for the same amount of protected memory compared to uniform, bit position, and access-frequency-aware selection of critical data. Further, CEF-aware FI reduces the FI time by $23,000\times$ with minimal loss of accuracy, and finds the failure rate of the MPA and individual bits. Finally, we study the impact of architectural design parameters on resilience and error mitigation overheads, and demonstrate the potential for designing resilient MPAs architectures using the CEF.

5.1 Architecture Overview

This section provides a brief overview of the microarchitecture of the Collision Detection Modules (CDM) studied in this chapter. This section focuses on collision detection modules for probabilistic roadmap-based motion planning approaches explained in Section 2.1.1.

5.1.1 Collision Detection Module

The collision detection step is the most time- and energy-consuming in motion planning and takes up to 99% of the total runtime on a CPU [18]. As collision detection must be performed in real-time with low latency to ensure safety in an environment with dynamic obstacles, MPAs typically dedicate more than 85% of their total area to accelerate collision detection [153, 172, 173], making the CDM the most vulnerable component in an MPA. An erroneous collision detection can potentially lead to a collision between the robot and an obstacle in its surroundings, making collision detection safety-critical. Many approaches have been proposed to accelerate collision detection on different hardware platforms. The architecture proposed by Murray et al. [171] uses specialized combinational circuits for a given motion set, but it is not reconfigurable to different motion sets at runtime. In contrast, GPU-based collision detection acceleration [18, 76] provides high flexibility, but it is not energy-efficient. Configurable collision detection hardware accelerators [8, 11, 153, 172, 173, 226, 278] provide a balance between flexibility and performance/energy-efficiency. We focus on the fault characterization of such reconfigurable CDMs of four MPAs, which we refer to as A1 [171], A2 [173], A3 [153], and A4 [278] throughout this paper.

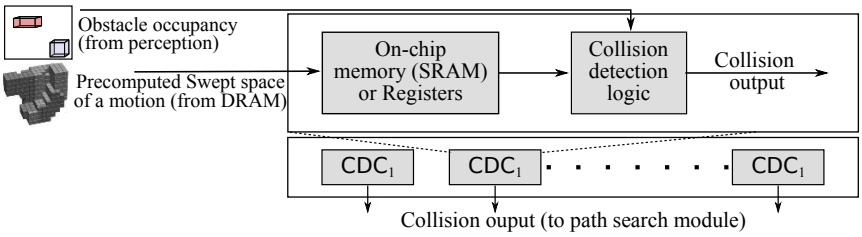


Figure 5.2: Architecture of a Collision Detection Module (CDM).

Figure 5.2 illustrates the architecture of a generic configurable CDM. The CDM consists of multiple collision detection circuits (CDCs) to check multiple motions for collision in parallel. Each CDC consists of on-chip memory/registers to store a motion's swept space, and exploits inter- and intra-query data reuse. The on-chip storage can be configured for different motions at runtime. Based on our synthesis experiments, the on-chip memory to store swept spaces constitutes more than 97% of the sequential elements and $\sim$ 50% of the total area in CDMs. Earlier work on FPGA-based accelerators [173] uses combinational logic to encode swept-space data. However, most programmable CDMs have a high fraction of sequential elements dedicated to on-chip storage for spatial data [153, 172, 173, 278]. We also verify this using RTL synthesis of these programmable CDMs (architecture explained below). Therefore, we focus on the on-chip memory for fault characterization, and do not consider the effect of soft errors in combinational logic in this work.

The key design consideration of CDMs is the geometric representation used to store swept spaces in the memory. Several representations for storing the 3D model have been proposed for motion planning, including polygonal meshes [5], bounding box hierarchies [84, 255], voxels [233], and octrees [117]. These approaches differ in storage requirements, representation accuracy, and computational complexity. All CDM hardware accelerators studied in this work use either grid-based or octree-based representations (explained in Section 5.2.1) as these representations are less compute- and memory-intensive. For both representations, the swept space of a motion is discretized into fixed-size cubes known as *voxels*. Voxelized swept space is then stored using a set of structures specific to the representation used. At runtime, a perception sensor module senses obstacle occupancy information, converts it to

Table 5.1: Accelerators studied. We list power and area for each accelerator from the paper and report suitable error mitigation and accelerator area overhead to protect on-chip memory. We use information gathered from our synthesis of RTL models about the fraction of CDM area and power consumed by the on-chip memory to calculate the overall CDM area and power overheads. ECC overheads do not include the area /power of decoder and encoder circuits.

Accelerator	Representation	Data reuse	#CDCs	Power (W)	Area (mm^2)	Storage elements	ECC area/power overhead	TMR area/power overhead
A1 [173, 233]	Voxel	Yes	32,768	N/A	N/A	Registers	20/16%	125/100%
A2 [173, 233]	Box	Yes	32,768	35	400	Registers	20/16%	125/100%
A3 [153]	Octree	No	128	0.47	1.7	SRAM	12/9%	100/75%
$A3_{scaled}$ [153]	Octree	Yes	32,768	121	450	SRAM	12/9%	100/75%
A4 [278]	Flattened Octree	Yes	32,768	20	-	SRAM/DRAM	12/12%	200/200%

voxels, and sends it to the CDCs. The CDCs perform collision detection for stored motions and send the output to the path search module. Note that the proposed metric CEF is applicable regardless of the geometric representation used.

5.1.2 Detailed Microarchitecture

Table 5.1 summarizes the four accelerators studied in this work and overall CDM area/power overheads for complete protection of on-chip storage using ECC and TMR error mitigation techniques.

A1 (Base accelerator): This architecture was proposed by Murray et al. [173], and is based on the earlier proposed accelerator for FPGAs [171]. A motion's swept space is stored in registers using the 3D Cartesian coordinates of each voxel in the swept space. Figure 5.4a and 5.4b show how a swept space is converted to voxels. The collision detection circuit compares the obstacle occupancy voxels with each voxel in the swept space to find if the motion is in collision with obstacles (Figure 5.3a).

A2 (Spatial locality-aware accelerator): This architecture, proposed by Murray et al. [173] and Sorin et al. [233], is an optimization of A1. There is a significant degree of spatial locality in the voxels in a swept space; hence, contiguous voxels

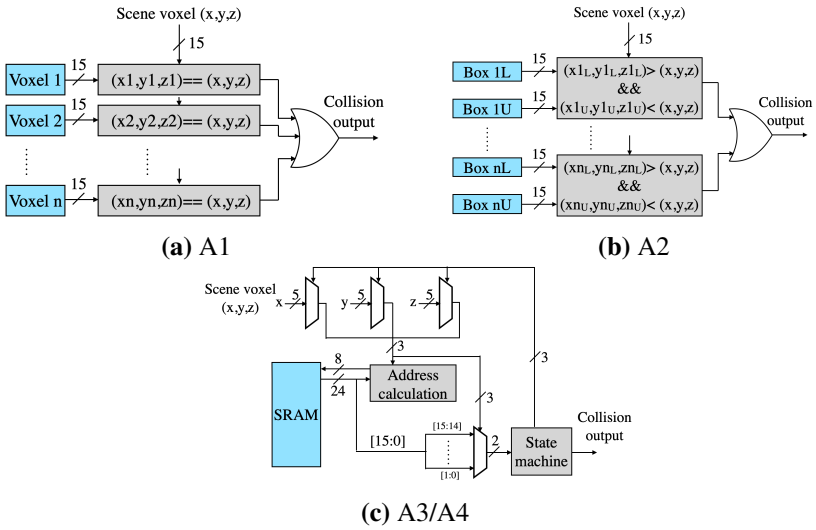


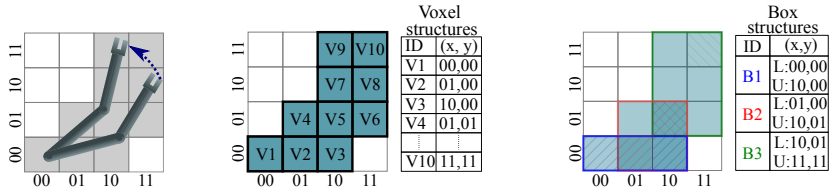
Figure 5.3: CDU architecture block diagram for different CDMs.

are merged into a larger box. A box can be represented by the coordinates of two diagonal voxels. Figure 5.4c gives an example. The CDC checks if an obstacle occupancy voxel is inside any of these boxes for collision detection (Figure 5.3b).

In A1 and A2, all on-chip registers are read in parallel by the collision detection logic instead of using a memory array with limited read ports. This design choice provides significant speedup. Thus all storage elements in A1 and A2 are treated as individual registers that can be protected using a latch protection technique (e.g., DMR or TMR).

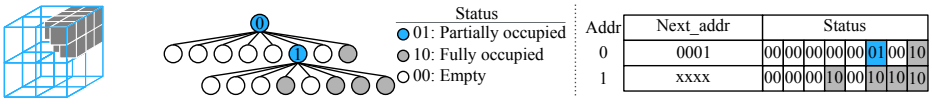
A3 (Octree-based accelerator): This architecture was proposed by Lian et al. (2018) [153], and uses the octree structure to store the motion's swept space (shown in Figure 5.5 and explained in Section 5.2). Collision detection is performed by traversing the tree to find if obstacle occupancy voxels overlap with the swept space. Figure 5.3c represents the architecture of CDC for A3.

The proposed design of A3 uses 128 CDCs, where motions in the motion set are processed for collision detection in batches. Hence, there is no inter-query on-chip data reuse, which results in significant DRAM bandwidth requirement [278]. For comparison, we also study a scaled-up version of A3, called $A3_{scaled}$, where the number of CDCs is equal to A1 and A2 ($32,768$), and on-chip data is reused across



(a) Swept space **(b)** Voxel (A1) **(c)** Box (A2)

Figure 5.4: Voxelization and voxel/box based representation example in 2D. (a) Represents a robot and its motion in 2D. Shaded voxels represent the voxelized swept space. (b) Swept space is stored using coordinates (x,y) of individual voxels. (c) Contiguous voxels are combined into boxes and stored using diagonal voxels' coordinates (striped voxels). L and U represent lower and upper diagonal voxels; darker regions represent overlapping of multiple boxes.



(a) **(b)**

Figure 5.5: Octree representation (figure reproduced from [153]). (a) represents the spatial division of the swept space, and (b) represents the corresponding Octree structure (A3).

multiple collision queries, reducing DRAM memory traffic.

A4 (Flattened octree-based accelerator): This architecture was proposed by Yang et al. [278], and proposes processing-in-memory for collision detection with a flattened octree-based representation of the swept space. In the flattened octree-based representation, multiple levels of the trees can be flattened in a single level. For example, if all the levels of a 5-level tree are flattened, the resultant tree consists of a single root node with $32,768$ children nodes, where each child node is 1 or 0 specifying occupancy of a single node. Such representation consumes more storage but facilitates efficient processing-in-memory, reducing data-movement overhead significantly.

5.2 Reliability Metric for Motion Planning

In this section we explain how erroneous collision-detection outcomes can lead to safety-critical events, then describe *Collision Exposure Factor* (CEF) in detail, and finally discuss how to apply CEF to building resilient MPAs. Specialized accelerators, such as a CDM, obtain efficiency by replacing long sequences of software instructions with specialized hardware. Such accelerators may perform computations under the supervision of a command processor via an ISA interface (e.g., Google's TPU [60]), and/or may start computation triggered by an event such as the arrival of a new frame of data in a buffer [41]. While an error originating in the accelerator could potentially propagate to the command processor and thereby result in erroneous operation (e.g., hang or system software crash) this paper focuses on Silent Data Corruption (SDC) within the CDM that can result in the robot colliding with an object.

Given a motion set and the current positions of obstacles, collision detection finds motions that may lead to collisions. A bit-flip can lead to erroneous collision detection. A false-positive outcome can lead to a poor (longer) path or the motion planner's inability to find a path, in which case the robot may get stuck. We perform fault injection experiments to measure the probability of a soft error resulting in a false-positive outcome; this gives an upper bound on the probability of the motion planner's inability to find a path due to soft errors. We find that the motion planning failure rate increases by up to 0.001% for 1 second/motion planning query [202] and 45nm technology node due to soft error induced false positive outcomes, which is significantly lower than the failure rate of probabilistic motion planning algorithms ($5-10\%$ [202]). Further, a simple mechanism such as using a watchdog timer to refresh the corrupted on-chip data can be employed to avoid such cases. We do not focus on false-positive outcomes in this work, as their impact on the motion planning failure rate is very low, and safety standards do not consider false-positives but only safety violations.

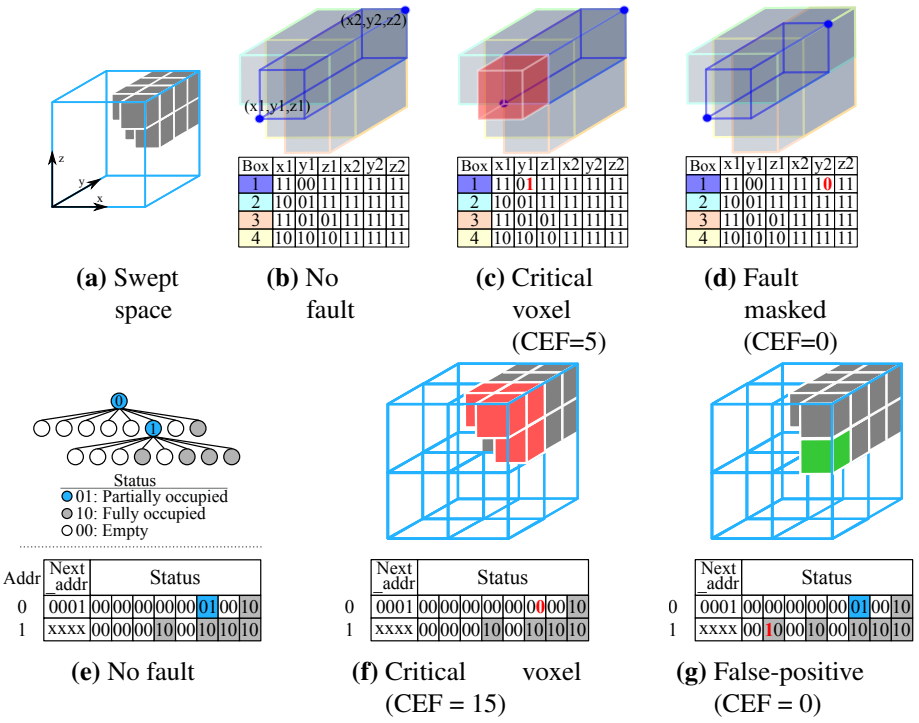
We define an SDC-C as an event involving a false-negative result when performing collision detection for a proposed motion and an object (i.e., a colliding motion is misidentified to be collision-free even though sensors detected an object the proposed motion would collide with if chosen during path search). Similar to AVF,

SDC-C probability gives a probability of raw error in a memory bit becoming an error in motion planning (i.e., false-negative collision detection). Since, in general, an obstacle might appear anywhere in the environment, a naive (but expensive) approach to estimate SDC-C probability is to simulate many sample environments, each with randomly placed obstacles. Below we consider a more efficient approach suitable, for example, during CDM architecture design.

5.2.1 Collision Exposure Factor (CEF)

To analyze the errors that can occur in a CDM circuit and their effect on SDC-C probability, we focus on bit changes that can lead to false-negative collision detection. Specifically, we consider the impact of a change in a single bit used to represent a portion of a motion's swept space. As mentioned in Section 5.1, swept spaces of the motions are stored in CDM memory or registers and used to find possible collisions with the obstacles. Each bit in the on-chip storage helps specify the bounds of some motion's swept space. We define the *critical space* of a bit as the region excluded from the swept space if that bit changes value due to a fault. This region represents a part of the swept space that can potentially lead to false-negative collision detection if an obstacle overlaps with the critical space and does not overlap with the remaining swept space. We then define the *collision exposure factor* (CEF) of a bit as the surface area of that bit's critical space exposed to obstacles. If the geometric representation uses voxelized space, the CEF can be normalized to the surface area of one face of a voxel. In our evaluation (Section 5.5.1), reported CEF values are unitless. The CEF values for different accelerators (A1-A4) are on the same scale as the same normalization factor is used.

Figure 5.6 illustrates the critical space of specific bits due to single-bit errors for different geometric representations. Figure 5.6a represents the swept space of a motion. Geometric representation methods convert the swept space to a set of *structures*, such as voxels (used in A1 described in Section 5.1), boxes (used in A2), or octree nodes (used in A3 and A4). Each of these structures is encoded into bits and stored in the on-chip storage of the CDM. For example, a voxel structure is stored using its coordinates, a box structure is stored using the coordinates of



(a) Swept space **(b)** No fault **(c)** Critical voxel (CEF=5) **(d)** Fault masked (CEF=0)

(e) No fault **(f)** Critical voxel (CEF = 15) **(g)** False-positive (CEF = 0)

Figure 5.6: Analyzing impact of bit flips in a CDM. (a) represents the swept space of a motion stored in CDM, (b)-(d) represent box-based CDM (A2), and (e)-(g) represent octree-based CDM (A3). CEF values are normalized to the surface area of one face of a voxel.

the diagonal voxels, and an octree node structure is stored using a custom data structure described below. A bit flip caused by a soft error would modify the space represented by the structure in different ways depending on the location of the bit in the structure and the geometric representation.

Figure 5.6b shows a box-based representation of the swept space, where four boxes are required to cover the swept space. Since space is divided into four voxels in x, y, and z directions, a total of six ($\log_2 4 \times 3$) bits are used to store a coordinate. The highlighted box is represented with coordinates $(x1, y1, z1)$ and $(x2, y2, z2)$ of the diagonal voxels, before any error is introduced. In Figure 5.6c, flipping a specific bit causes the value of $y1$ to increase by one voxel. As the voxel highlighted in red is now excluded from the box, and no other box covers the voxel, it becomes

part of the bit's critical space. A soft error in this bit results in the exclusion of the critical space from swept space; hence the collision detection circuit fails to detect a collision if an obstacle occupies this critical space, leading to an SDC-C. In contrast, in Figure 5.6d, a bit flip causes $y2$ to decrease, but this bit flip is masked and does not impact the CDM's output as other boxes cover the voxels exposed.

Figure 5.6e shows the octree representation of the same swept space. In this representation, the root node and all partially occupied nodes are stored in memory using a tree data structure. Each node in the tree contains two fields: "`status`" and "`next_addr`". The `status` field contains an entry for each octant within the node indicating whether that octant is empty, partially or fully occupied. Only partially occupied octants are further divided. The `next_addr` field contains the start address of an array holding the resulting children nodes. For example, in Figure 5.6e, the node stored in memory at address 0 contains only one partially occupied octant, and the node containing information about it is stored at address 1. In Figure 5.6f, a bit flip in the node at address 0 modifies the status of a particular octant to be "fully empty", thereby adding all the voxels in that octant to the critical space. In Figure 5.6g, a bit flip in the node at address 1 modifies the status of an octant to "fully occupied", resulting in a false-positive rather than a false-negative. In this case, motions that would not lead to a collision may be disallowed. However, this false-positive outcome does not result in potential for a collision (an SDC-C). Therefore, this voxel (highlighted in green) is not a part of the critical space.

For a given erroneous bit, it is more likely that an obstacle occupies its critical space and results in an SDC-C as the exposed surface area of this critical space increases. Hence, intuitively, the SDC-C probability of a bit increases with its CEF value (Section 5.5.1). Note that the CEF of a bit is *independent of the position of the obstacles in the environment*. The CEF captures the probability of obstacles appearing in critical space and decouples the effect of a soft-error and exact position of obstacles on SDC-C probability. While the CEF definition assumes a uniform distribution of obstacles, it can also be extended to nonuniform distributions. For example, the CEF value can be scaled by the estimated probability of obstacles occupying the critical space for a nonuniform distribution of obstacles in the environment.

One approach is to use the volume of the critical space (CS_volume) as a

reliability metric. However, the exposed surface area gives a measure of the area of critical space through which obstacles can overlap only with the critical space and not the rest of the swept space, leading to an SDC-C. In contrast, CS_volume does not differentiate between the surface area exposed to obstacles and the surface area touching the remaining swept space. Hence CEF performs better than CS_volume as a reliability metric (Section 5.5.2). For a given motion set of the robot and

Algorithm 2 CEF measurement (Phase 1 FI)

Input: Motion_set, Swept_data, Swept_voxels, CDM;
Output: bit_info = {bitID: (CEF, Critical_space)};

```
 1: for Motion ∈ Motion_set do
 2:     bits = Swept_data(Motion)
 3:     voxels = Swept_voxels(Motion)
 4:     for b ∈ bits do
 5:         Critical_space = ∅
 6:         Collision_vector = FI(CDM, bits, voxels, b)
 7:         for (v, collision) ∈ (voxels, Collision_vector) do
 8:             if ¬ collision then
 9:                 Critical_space = Critical_space ∪ v
10:             end if
11:         end for
12:         CEF = Calculate_CEF (Critical_space, voxels)
13:         bit_info[b]= (CEF, Critical_space)
14:     end for
15: end for
```

accelerator, the CEF of all the bits can be calculated using Algorithm 2. The algorithm works by considering each possible motion in turn (Line 1). For a given motion, `Swept_data` returns the storage bits used to represent its swept space (Line 2), and `Swept_voxel` returns the voxels in the swept space of that motion (Line 3). The loop between Line 4 and 14 considers each bit in the `Swept_data`. The CDM takes precomputed swept space (i.e., `bits`) and obstacle occupancy voxels as inputs and performs collision detection (Figure 5.2). The storage elements of the CDM are initialized with `bits`. To find the critical space an FI run is performed setting `voxels` as the obstacle occupancy voxels input to the CDM (Line 6). Specifically, a fault is injected into a low-level (e.g., RTL model or microarchitectural simulator) model of the CDM at bit `b` (Line 6). The CDM outputs a `Collision_vector` containing collision detection output for each voxel in `voxels`. To find the critical space, each bit of the `Collision_vector` is checked (Lines 7-11). For an error-free run, the CDM detects collision for all the voxels in the swept space (i.e., `voxels`). However, for an FI run, a voxel `v`

is added to the `Critical_space` if the CDM does not detect a collision (Line 8 and 9). `CEF` is then calculated by measuring the exposed surface area of the `Critical_space` (Line 12). Note that the CEF is obtained without the need to consider a potentially unbounded set object placements. Thus the number of FI runs to measure CEF using Algorithm 2 is orders of magnitude lower than exhaustive FI in which for each bit, multiple FI runs need to be performed with a large number of environment scenarios.

As presented above, Algorithm 2 assumes an RTL or architecture model for the CDM. We note that the resulting reliance upon fault injection to determine CEF could be avoided provided an analytical model is available to compute the critical space on Lines 5 to 12. Such a model could be used to analyze CEF without performing FI prior to the development of RTL model or architecture simulators. However, we use FI to measure CEF, so the proposed Algorithm 2 can be used across different accelerators. Algorithm 2 is proposed for CDMs that use a voxelized representation of spatial data and output collision decisions (A1-A4). The majority of the collision detection acceleration approaches for robotics use voxelized representation as it consumes less memory and computational resources compared to polygonal mesh-based representation [57]. However, the concept behind CEF is applicable regardless of the underlying design parameters, and Algorithm 2 to measure the CEF needs can be modified for a different CDM. For example, for a triangle meshes-based representation of swept space, Line 5-12 can be replaced by a geometry-based calculation of the exposed surface.

5.2.2 CEF-aware Error Mitigation

For a fixed budget of area/power overhead, selective protection of the most vulnerable bits provides the optimal reduction in the failure rate. Selective error mitigation in storage structures can be implemented by reliability-aware data placement to partially protected memory. However, this requires efficient ways to identify vulnerable data as the data can be generated/modified at runtime. We find that the CEF of a bit gives a measure of its impact on the SDC-C probability of the CDM (Section 5.5.1). Therefore, we can use information about the CEF of each bit in an input design to selectively apply error mitigation techniques such as ECC, DMR/TMR, and strike

suppression. This reduces the cost of providing resilience compared to blanket protection of all the bits.

Different structures such as voxel (A1), box (A2), and octree node (A3 and A4) are stored in the on-chip storage of the CDM depending upon the geometric representation method used. CEF-aware error mitigation is performed by placing structures with higher CEF in the protected storage regions. In our evaluation, the sum of CEFs over all the bits in a structure is used as its CEF. Because CEF of a bit gives a measure of its SDC-C probability, CEF-aware data placement results in a higher reduction of the overall failure rate for a given fraction of protected storage compared to other heuristics (Section 5.5.2). The CEF measurement is done offline or outside of the critical path using Algorithm 2 after motion set generation.

Selective error mitigation can also be used to guide the accelerator design process. At the accelerator design stage, the designer can perform CEF-aware FI (Section 5.3.2) for a set of target robots, motion sets, and the expected number of obstacles in the environment to find the distribution of CEF, SDC-C probability of bits, and the failure rate of the system without any error mitigation. Based on this information, a designer can determine the required fraction of protected memory for a given failure rate requirement or achievable failure rate for a given area/power budget for error mitigation.

5.3 CEF-aware Reliability Characterization

SDC-C probability and the failure rate measurements are important to find the error mitigation requirements for certifiable safety. Exhaustive FI to find the SDC-C probability of all the bits takes a long time (Section 2.3.3). In this section, we first explain the fault model and demonstrate how to use the CEF to enable a hierarchical fault analysis methodology that reduces the number of FI runs.

5.3.1 Fault Model

In the CDMs studied in this work, on-chip SRAM/latches that store the swept space of motions account for 97% of the on-chip sequential elements (based on our synthesis results). Our fault model assumes transient single-bit faults in this SRAM/latches. The faults are uniformly distributed in time and space. Only

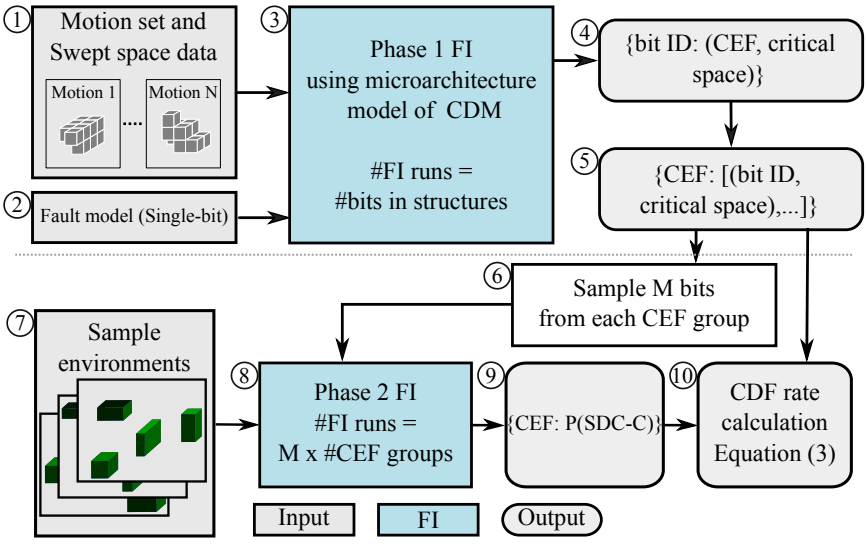


Figure 5.7: CEF-aware FI

3% of on-chip sequential elements are used for meta-data (e.g., counters, address register, and finite state machine). These microarchitecture-specific registers are not subjected to CEF in our experiments as they can be characterized and protected with low overheads using latch hardening techniques such as strike suppression or redundant node [219].

5.3.2 CEF-aware FI

We propose a two-phase CEF-aware FI (shown in Figure 5.7), a technique to speed up FI for MPAs.

Phase 1: CEF measurement (Environment independent)

For a given motion set ① and fault model ②, Phase 1 performs microarchitecture- or RTL-level FI ③ to find the CEF and critical space of each bit in the swept space data. Algorithm 2 is used for this phase. The number of FI runs for this phase is limited to the number of bits in swept space data of all the motions in the robot's motion set. The CEF and critical space of all bits are stored to be used in the next phase ④. Note that environmental information, such as the position of

obstacles or the robot, is not needed for this phase, as CEF does not depend upon the environment.

Phase 2: SDC-C probability measurement

A bit flip might lead to an SDC-C depending upon the position of obstacles as shown in Section 5.2. Since for a dynamic environment, obstacles can appear anywhere in the space, a large number of FI runs with random environmental scenarios are required to measure the SDC-C probability of a bit with statistical significance. In our experiments, the representative environment scenarios are generated using apriori information about the environment, such as the distribution and average number/size of the obstacles. In Phase 2, we use the CEF information gathered in Phase 1 to speed up the fault analysis. As mentioned, there is a strong (positive) correlation between the CEF and SDC-C probability (Section 5.5.1). Thus, we can speed up the FI experiments many-fold by performing FI for only a subset of bits with a given CEF value to measure the approximate SDC-C probability for all bits with the same CEF. The CEF information gathered in Phase 1 is used to group bits with equal CEF values together ⑤. Then, for each CEF value, M bits are selected at random ⑥. Finally, using multiple sample environment scenarios ⑦ and FI simulations ⑧, the SDC-C probability for each CEF value is measured ⑨. Note that M is a tunable parameter in the above heuristic. We use the analytical model in Leveugle et al. [145] to determine the value of M to measure SDC-C probability with the required confidence level and error margin. The CEF-aware sampling, while being faster than exhaustive FI, may introduce inaccuracies due to the approximations it makes. However, the accuracy can be increased by increasing the value of M - we evaluate this trade-off in Section 5.5.1.

To speed up the FI simulation, we further exploit the fault propagation in the studied CDMs. All the accelerators studied in this work use a geometric representation that divides the space into voxels. For a given motion and environment scenario, a non-empty subset of swept space and obstacle occupancy voxels signify potential collision. Thus the effect of a bit flip can be captured by storing the erroneous swept space voxels of the corresponding motion, instead of performing slow microarchitectural and RTL simulations for multiple environmental scenarios.

Simple set operations can be used to find the SDC-C probability. For a given bit flip, an FI simulation is used to find the erroneous swept space (i.e., set of voxels) using Algorithm 2. Line 3 of the Algorithm 2 is modified to use the set of all environment voxels as `voxels`. Similarly, Lines 8 and 9 are modified to measure the erroneous swept space (i.e., colliding voxels). For each environment scenario, the subsets of obstacle occupancy voxels and erroneous and error-free swept space are measured. If the erroneous subset is empty, but the error-free subset is non-empty, the bit flip will result in an SDC-C. The same strategy is used for exhaustive, statistical, and CEF-aware FI. While we focus on FI to measure CEF and SDC-C probability, we believe that an analytical model can be used to replace the multiple runs and find the relation between critical space and SDC-C probability. We defer this to future work.

Equation 5.1 is used to calculate the SDC-C probability of a CDM. Here bits_{CDM} is the set of all bits stored in the CDM's on-chip memory. CEF(x) is the CEF value of bit x, and $\text{P}(\text{SDC-C}_{\text{CEF}(x)})$ is the SDC-C probability for the CEF value CEF(x).

$$\text{P}(\text{SDC-C}_{\text{CDM}}) \approx \frac{1}{|\text{bits}_{\text{CDM}}|} \times \left[\sum_{x \in \text{bits}_{\text{CDM}}} \text{P}(\text{SDC-C}_{\text{CEF}(x)}) \right] \tag{5.1}$$

Table 5.2 compares the different phases of CEF-aware FI with exhaustive and uniform random statistical FI approaches for CDM fault characterization and error mitigation. We also use CEF and CEF-aware FI to analyze and compare the effects of microarchitectural design parameters of the CDM, and to derive the principles of *resilience-aware* MPA design (Section 5.6).

5.3.3 Collision Detection Failure (CDF) Rate Calculation

As mentioned in Section 2.3.2, safety standards provide an upper bound on the frequency of dangerous failures (e.g., collision in motion planning). We combine Equation equation 2.1 and Equation equation 5.1 to measure the Collision Detection Failure rate per billion hours (CDF) for CDMs as below:

$$\text{CDF}_{\text{CDM}} = |\text{bits}_{\text{CDM}}| \times \text{P}(\text{SDC-C}_{\text{CDM}}) \times \text{FIT}_{\text{Raw}} \times \alpha \tag{5.2}$$

Table 5.2: Comparison of different fault injection (FI) approaches.

	Exhaustive FI	Uniform random statistical FI	CEF-aware FI	
			Phase 1 (CEF measurement)	Phase 1 + Phase 2 (SDC-C measurement)
Description	Performs FI on all combinations of bits and environment scenarios	Samples a few combinations of bits and environment scenarios to perform FI	Performs environment-agnostic FI to measure CEF of all the bits (Algorithm 2)	Samples a few combinations of environmental scenarios and bits for each CEF group and performs FI
Number of FI runs	10^{10}	7×10^6	10^6	7×10^6
FI time	24,000 hours	2-4 hours	1-2 hours	2-4 hours
Measures overall failure rate	Y	Y	N	Y(2.5% error)
Finds vulnerable bits for selective error mitigation	Y	N	Y(Approx.)	Y(Approx.)
Measures SDC probability of different bits	Y	N	N	Y(Approx.)

Here, the parameter α is set to $N/2$ for CDMs with inter-query data reuse, else it is set to 1, as explained below. In the MPAs we study, the on-chip data is typically reused across multiple executions of the same application [93, 173, 233, 278] to reduce the DRAM-bandwidth requirement and data movement. In such a case, a bit-flip due to a soft error will persist in the buffer and affect multiple executions, until the buffers are reloaded, and the overall CDF rate of the system increases due to data reuse. This is similar to bit-flips in configuration memory of FPGA [182], where a bit-flip due to single-event upset persists and affects the application output until the configuration memory is refreshed. A1, A2, $A3_{scaled}$, and A4 accelerators reuse on-chip data across N (where $N >> 1$) collision detection queries. A soft error can occur in any cycle during a collision detection query; however, the bit will remain erroneous from the start (i.e., cycle 0) for subsequent collision detection queries until the on-chip data is refreshed. Thus the average value of the number of collision detection queries for which a bit will be erroneous from starting is approximately $N/2$, assuming all collision queries take the same time. Hence, to measure the CDF rate for such CDMs, faults are injected at cycle 0 in Phase 2, and the measured failure rate is scaled by $\alpha = N/2$ to account for data reuse. Whereas in A3, the on-chip data is refreshed for every collision detection query. For CDMs without inter-query data reuse, α is set to 1, and the fault sites are uniformly distributed in

time in Phase 2.

5.4 Experimental Methodology

Table 5.3 summarizes the robots used in our experiments. These robotic arms are representative of widely used industrial robots [169, 209], and are also included in larger humanoid robots [2]. Because we did not have access to the real robots, we use the Klampt [100] software simulator to simulate the robot's movements and the environment- this has also been used in prior work [99, 171, 233].

The environment size for a robot is determined by its reach, and the environment is discretized into a grid of $32 \times 32 \times 32$ voxels. For each robot, we generate a motion set with $16,384$ poses and $32,768$ motions, using Leven and Hutchinson's strategy [144], similar to prior work [171]. We use uniform motion sets in all our experiments, i.e., poses are distributed uniformly in the C-space. The uniform distribution of poses is assumed in the absence of any prior information about the robot's task and fixed obstacles in the environment. However, this information can be used to build a motion set with a nonuniform distribution of poses, where more poses are concentrated in the region of interest [253]. We also evaluate CEF on a nonuniform motion set in Section 5.5.1.

To measure the SDC-C probability, we perform FI with a set of $10,000$ random environment scenarios. Each sample environment contains $3-12$ cuboid-shaped randomly placed obstacles, and the length/height/width of each obstacle is $5-20$cm, which is consistent with other work on motion planning and collision detection [153, 171]. We generate four sets of environments to study the effect of the number of obstacles on the SDC-C probability (Section 5.5.1). We use the label Dx ($x \in [1,2,3,4]$) to represent a set of $10,000$ environment scenarios, where obstacles occupy an average x% volume of the environment. The average number of obstacles

Table 5.3: Robots used for fault characterization.

Robot	Degrees of freedom	Reach in one direction	Mechanical power (W)
Kinova Jaco2 [132]	7	90 cm	25
Programmable Universal Machine for Assembly (PUMA) 761 [269]	6	150 cm	30
AL5D [159]	4	27 cm	12

increases with the value x.

5.4.1 RTL Models and Synthesis

We built microarchitectural simulators and RTL models of all CDMs ourselves as there was no existing simulator. We use Verilog to build the RTL models and implement the combinational logic, control logic, and on-chip memory as described in the prior work on the accelerators we studied in this work [153, 173, 233, 278]. CDMs studied in this work consist of multiple (128-32768) CDUs. Hence, though the overall area of a CDM is high, the design of the basic building block, CDU, is simple and can be implemented accurately ($100-200$ lines of Verilog code). Further, we implement testbenches to verify the functionality and timing of the RTL models. We use a sample motion set, and perform collision detection for $10,000$ environmental scenarios for functional verification. We further measure the timing of collision detection operation in terms of the number of cycles and validate with the reported data in the prior work [153, 173, 233, 278].

We synthesized our RTL models using the Synopsys Design Compiler [1] and the OpenRAM Memory Compiler [91] to estimate the area and power of storage elements in CDMs at 45nm technology (FreePDK45 design library [238]). Because we are interested in the relative area and power consumption of different storage elements and combinational circuits, the technology node's choice should not significantly impact the results. We build the microarchitectural simulators with Python. We model all sequential elements accurately in our microarchitectural model, and verify the collision detection output of these models for $10,000$ environmental scenarios.

5.4.2 Fault Injection (FI)

We use microarchitectural simulators for FI, as RTL-level FI was very slow. We use Dell EMC R440 CPU nodes. However, to validate the FI accuracy of our microarchitectural simulators, we perform $10,000$ FI runs (uniformly distributed in space and time) on the microarchitectural and RTL models. For RTL-level FI, we use Cadence Incisive Functional Safety Simulator [30]. For microarchitectural FI, a fault is injected, and collision detection is performed. Figure 5.8 shows that there

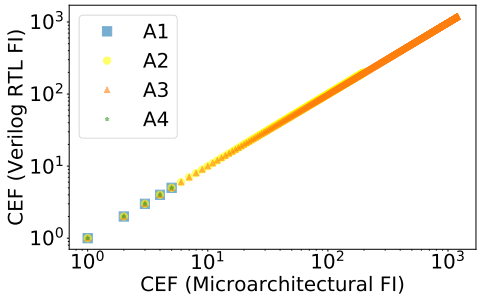


Figure 5.8: CEF values measured by microarchitectural FI and Verilog RTL-level FI for different CDMs.

is 100% correlation between the CEF values measured with microarchitectural FI and RTL-level FI across all four accelerators (A1-A4).

For CEF-aware FI, we determine the value of M (number of samples) per CEF group required to measure the SDC-C probability with 95% confidence level and 2.5% error margin [145]. The SDC-C probability and population size of the highest CEF group are used to determine the value of M, as these bits contribute the most to the overall SDC-C probability. In our experiments, the value of M is 2×10^6, 2×10^5, 15×10^3, and 2×10^6 for A1, A2, A3 and A4 respectively. We represent soft errors as single-bit flips in hardware registers, which is consistent with most other papers studying the effects of soft errors [9, 39, 67, 256]. While we focus on single-bit flips, Equation equation 5.1 can be extended to accommodate a multi-bit fault model. We defer this to future work.

5.4.3 CDF Rate Calculation

To calculate the CDF rate of the accelerators, we use Equation equation 5.2, where N is the expected number of executions of the application before the bits are reloaded in on-chip storage, and its value depends on the accelerator's architecture and deployment. Reloading of data can be overlapped with collision detection to hide the DRAM accesses latency. We set the value of $N = 3600$ for the accelerators with inter-query data reuse (A1, A2, $A3_{scaled}$, and A4), as the power overhead of DRAM accesses for refreshing data after 3600 collision detection queries is within 1%. We use $FIT_{Raw} = 177$ FIT/Mb (for 45nm CMOS [7]) in Equation equation 5.2. Note

that while the choice of N and FIT_{Raw} affect the absolute values of the CDF rate, they affect neither our fault characterization nor CEF-aware error mitigation.

5.5 Evaluation

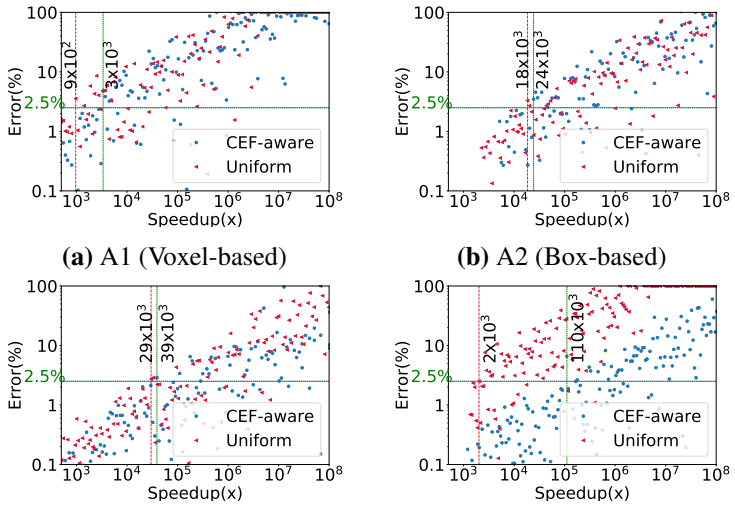
In this section we present our results for fault characterization (Section 5.5.1) and error mitigation (Section 5.5.2) using CEF.

5.5.1 Fault Characterization

CEF-aware FI

In Section 5.3, we propose two-phase CEF-aware FI to reduce the number of FI runs. In CEF-aware FI, instead of performing FI for all the bits for multiple environment scenarios, we sample a subset from a group of bits with the same CEF value. This sampling introduces inaccuracies in the measured SDC-C probability and CDF rates compared to exhaustive FI. Figure 5.9 shows the speedup versus error of the calculated CDF rate for CEF-aware FI and uniform statistical FI. The speedup is the ratio of the number of exhaustive FI runs to that of CEF-aware FI or uniform FI.

As can be seen from the figure, on average, CEF-aware FI achieves $23,000\times$ speedup over exhaustive FI (geometric mean) with 2.5% error margin. The vertical line represents the speedup for error less than 2.5%. The error margin can be reduced further at the cost of more FI trials (Section 5.3). Uniform statistical FI exhibits a similar speedup as CEF-aware FI over exhaustive FI. Note however that uniform statistical FI cannot be used to find vulnerable bits for selective error mitigation (Table 5.2). The speedup of CEF-aware FI over exhaustive FI is due to two reasons: (1) a significant fraction of bits have CEF equal to 0, and hence have very low or 0 SDC-C probability. CEF-aware FI ignores these bits for FI, (2) Only a few bits have high CEF and SDC-C probability, and hence contribute the most to overall SDC-C probability. CEF-aware FI segregates such bits and requires fewer samples to measure SDC-C. Phase 1 of CEF-aware FI consumes 1, 1, 2, and 1 CPU hours for A1, A2, A3, and A4, respectively. Phase 2 of CEF-aware FI consumes less than 2 CPU hours for all accelerators. In contrast, as per our experiments, exhaustive FI takes $24,000$, $18,000$, $22,000$, and $20,000$ CPU hours for A1, A2, A3, and A4,



(a) A1 (Voxel-based)

(b) A2 (Box-based)

(c) A3 (Octree-based)

(d) A4 (Flattened octree-based)

Figure 5.9: Error versus speedup for different FI approaches.

respectively.

Evaluation of reliability metric

As discussed in Section 5.3, we propose the CEF as a reliability metric for the MPA's bits. We study the relationship between the CEF and the SDC-C probability of different bits to demonstrate the validity of CEF metric. The SDC-C probability of entire CDM is calculated using SDC-C probability of different CEF groups in Equation equation 5.1. We use the benchmarks [D1-D4] (Section 5.4). Figure 5.10 shows the SDC-C probability of bits with different CEF values for different accelerators, robots, and benchmarks. The CEF values are on the same scale for different accelerators for a given robot. A3 has a higher range of CEF values than other accelerators, which is also reflected in higher SDC-C values. Note that for A3, there is dispersion in SDC-C probability for bits with high CEF values. This occurs as a fault in a bit in the `next_addr` field adds a large number of voxels in the represented swept space (green voxels in Figure 5.6g), which leads to positive collision detection between obstacles and erroneous expanded swept space, and reduces SDC-C probability for some bits with high CEF value. For all the benchmarks, the

SDC-C probability of a bit increases as its CEF increases, except for a few bits in A3. Note that as the obstacle occupancy density increases, the SDC-C probability also increases, as there are higher chances that the soft error in a CDM will result in a collision. Thus, even though the relation between SDC-C probability and CEF is not linear, a monotonically non-decreasing relation demonstrates the validity of the CEF as a reliability factor. *The SDC-C probability of bits is thus strongly positively correlated with their CEF values across benchmarks, which signifies that the CEF can be used as a reliability metric for MPAs.*

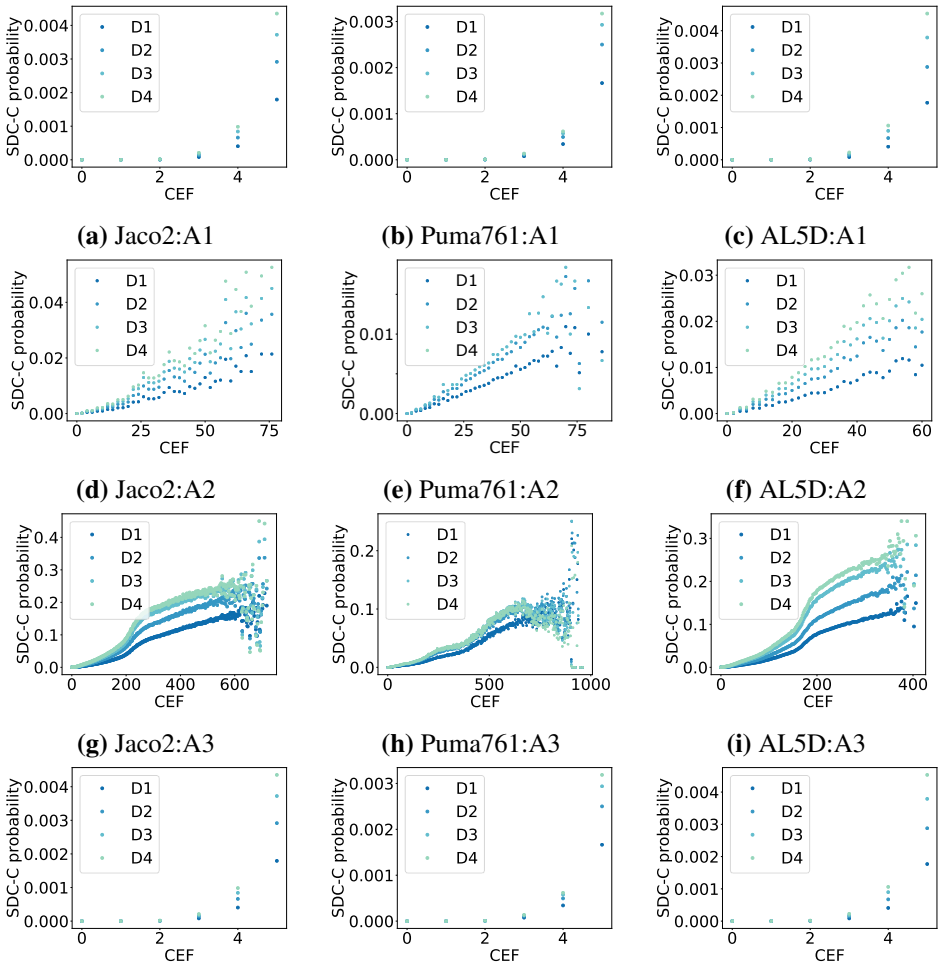
Nonuniform motion set We also validate the use of CEF as a reliability metric for a nonuniform motion set. In such a motion set, the motions are nonuniformly sampled in space. Figure 5.11 shows SDC-C probability versus CEF of bits for nonuniform motion sets. As shown in the figure, there is strong correlation between the SDC-C probability and CEF of bits, showing the validity of CEF for nonuniform motion sets.

Characterization of CEF

To further understand the contribution of individual bits to the overall SDC-C probability, we group the bits according to their CEFs, and plot the cumulative distribution of bits in the CDM. Figure 5.12 shows that for all four CDMs, there is a high degree of asymmetry in the distribution of bits according to the CEF. For A1 and A4, 20% and 99% of the total bits have CEF equal to 0, respectively, and do not need to be protected. On the other hand, for A2, only 20% of the total bits have CEF greater than 6 and significantly contribute to the overall SDC-C probability. Thus, protecting only 20% bits can reduce the CDF rate by 60%, as we show in Section 5.5.2. Similarly, in A3, only 15% of the total bits have a CEF greater than 10. We further examine the CEF distribution asymmetry for each accelerator.

A1: In A1, each structure/variable stores a voxel using its coordinates to represent a part of the swept space. A soft error in any bit of the coordinate will result in misrepresentation of that voxel, and so bit position does not affect CEF range (Figure 5.13a).

A2: The effect of faults in different bits of coordinates is shown in Figure 5.13b.



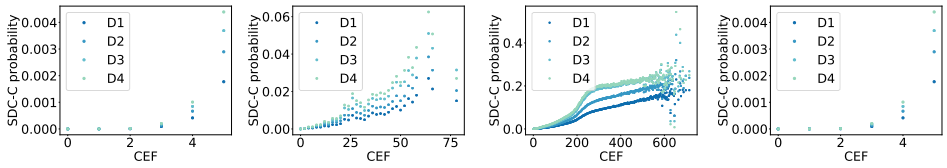
(a) Jaco2:A1 **(b)** Puma761:A1 **(c)** AL5D:A1

(d) Jaco2:A2 **(e)** Puma761:A2 **(f)** AL5D:A2

(g) Jaco2:A3 **(h)** Puma761:A3 **(i)** AL5D:A3

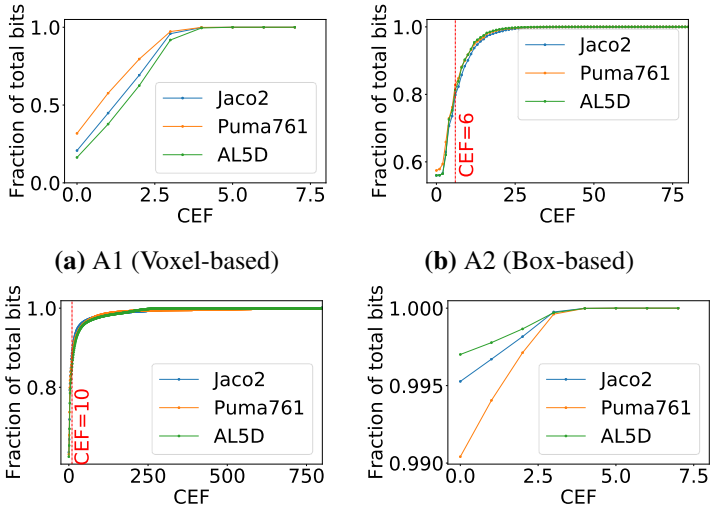
(j) Jaco:A4 **(k)** Puma761:A4 **(l)** AL5D:A4

Figure 5.10: SDC-C probability versus CEF of accelerator A1-A4 for Jaco2, AL5D, and Puma761 robot on benchmarks D1-D4. Note that different plots have significantly varying ranges for the vertical axes.



(a) A1 **(b)** A2 **(c)** A3 **(d)** A4

Figure 5.11: SDC-C probability versus CEF of accelerator A1, A2, A3, and A4 for Jaco2 for a nonuniform motion set. Note that different plots have significantly differing ranges for the vertical axes.



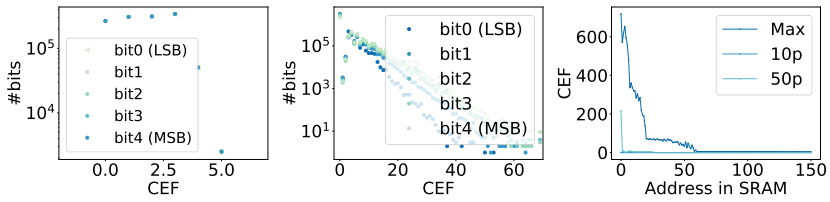
(a) A1 (Voxel-based) **(b)** A2 (Box-based)

(c) A3 (Octree-based) **(d)** A4 (Flattened octree-based)

Figure 5.12: Cumulative distribution of CEF of all bits. The vertical axis starts from 0.99 for A4 as more than 99% of the bits have CEF equal to 0.

The CEF range of bits decreases from the most significant bit (MSB) to the least significant bit (LSB), as a fault in the MSB is likely to result in a higher change in the box size represented by a pair of coordinates than LSB. Thus, the number of bits with high CEF values increases from LSB to MSB. For example, the ratio between the number of MSBs ($bit4$) and the number of LSBs ($bit0$) with CEF $= 20$ is 1000.

A3: We calculate the range of CEF for bits for different SRAM addresses, as shown in Figure 5.13c. All nodes in the octree for a motion's swept space are stored



(a) A1 **(b)** A2 **(c)** A3

Figure 5.13: CEF characterization of bit position/SRAM address.

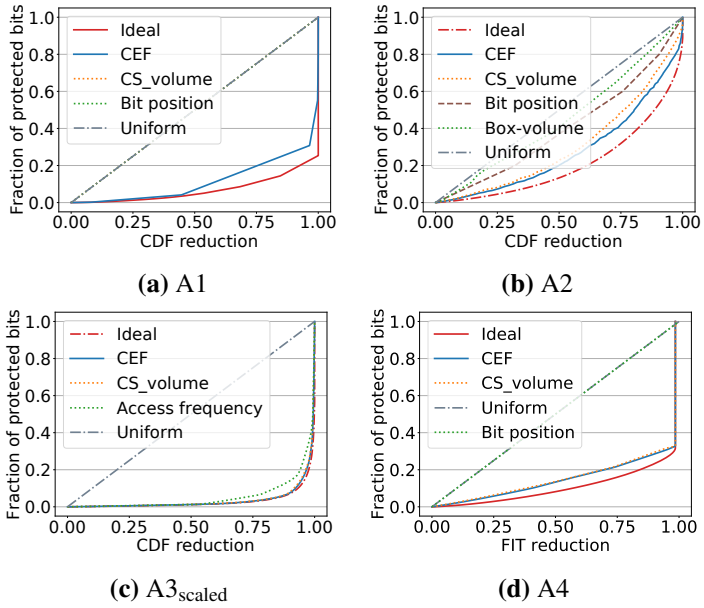
in the contiguous address space of the SRAM; the nodes closer to the octree's root node that divide the 3D space at a coarser granularity are stored in the lower address range of SRAM. As can be seen, the CEFs of bits with lower SRAM addresses are much higher than those with higher SRAM addresses. For example, the average CEF of bits in SRAM address 0 is 9× that of the average CEF of bits in SRAM address 20.

A4: A flattened-octree consists of a node for each voxel in the environment. The swept space of a motion in the motion set is a small fraction of the entire environment. Thus, most of the nodes in the flattened octree are empty, and only a small fraction of nodes are occupied. An error in an empty node representation results in false-positive and hence has CEF equal to zero. Due to this, more than 99% of the bits have CEF equal to 0 for A4.

In summary, for all four accelerators, the distribution of the bits as per CEF is highly asymmetric. Hence, the CEF metric facilitates finding the most SDC-C-prone bits in the circuit.

5.5.2 Error Mitigation Techniques

As described in Section 5.2.2, we propose a CEF-aware selective error mitigation technique. We first evaluate the CEF-aware selection of structures for reliability-aware data placement at deployment time. We further evaluate the area savings achieved by the proposed techniques for accelerators A1, A2, $A3_{scaled}$, and A4 for different SIL targets at design time. Note that A3's CDF rate (0.1) is well below the highest SIL requirement, and hence it does not need error mitigation.



(a) A1 **(b)** A2

(c) $A3_{scaled}$ **(d)** A4

Figure 5.14: Comparison of different selection criteria for selective error mitigation in A1, A2, $A3_{scaled}$, and A4. (a)-(d) compare the fraction of protected bits versus CDF rate reduction for different CDMs. Overall, CEF-aware selection results in the highest CDF rate reduction for the same amount of protected bits. There is a significant overlap between `Bit position`, `Uniform`, and `CS_Volume` for A1. There is a significant overlap between `Bit position` and `Uniform`, and `CS_Volume` and `CEF` for A4.

CEF-aware selective error mitigation

Figure 5.14 compares the CDF rate reduction achieved by different selection criteria for the same fraction of protected memory. We assume that protecting a bit reduces its SDC-C probability to 0 as our aim is to compare different heuristics for selecting the bits to be protected, which is independent of the underlying error mitigation technique. We also compare with another intuitive approach to define the reliability metric that uses the volume of the critical space (CS_volume) instead of the exposed surface area. We further discuss the results for A1, A2, $A3_{scaled}$, and A4.

A1: We compare our approach with ideal, uniform, bit position, and CS_volume-aware error mitigation. Bit position does not require FI and can be used as a proxy

for the vulnerability factor, where the vulnerability of the bits decreases from the MSB to the LSB. Bit position has been previously proposed for selective error mitigation in DNN accelerators [147]. Figure 5.14a shows the fraction of protected bits versus the CDF rate reduction curve for CEF-aware selection and the other heuristics for A1. The CEF-aware selection of bits results in 52.35× reduction in the CDF rate on average (geometric mean) for the same amount of protected bits compared to CS_volume-aware, bit position-aware, and uniform selection. The CDF rate reduction for CEF-aware error mitigation is only 1.60× lower than the ideal CDF rate reduction that can be achieved by exhaustive FI.

A2: We compare CEF with ideal, uniform, CS_volume, bit position, and box-volume. For A2, a structure represents a box; the volume of the box can be used as the vulnerability factor of the structure. *Box-volume* is an application-specific heuristic that does not require fault characterization. Figure 5.14b shows the fraction of protected bits versus the CDF rate reduction for different selection heuristics. The CEF-aware selection of bits results in average 1.25×, 1.76×, 2.10×, and 2.46× lower CDF rate than CS_volume, bit position, box volume, and uniform selection for the same amount of protected bits. The CDF rate reduction for CEF-aware error mitigation is only 1.66× lower than the ideal CDF rate reduction achieved by exhaustive FI.

Intuitively, one may expect bit position to provide higher benefits than CEF, as typically MSBs are more critical. However, we find that CEF provides a higher reduction in the CDF rate than the bit position, due to two reasons. First, though the bit position captures the failure probability of bits within a structure, it does not capture the relative failure probabilities across different structures. For example, in A2, different boxes cover different numbers of voxels x in the swept space that is not covered by other boxes; these voxels contribute to the critical space. The CEF increases as the x value increases, and so structures with higher x have higher failure probabilities, which is not captured by bit position. Second, the bit position does not consider whether the error will lead to a false-negative or a false-positive. Similarly, box-volume does not necessarily capture the number of critical voxels x. In contrast, the CEF captures the relative failure impact of all the structures, and hence provides a higher CDF rate reduction.

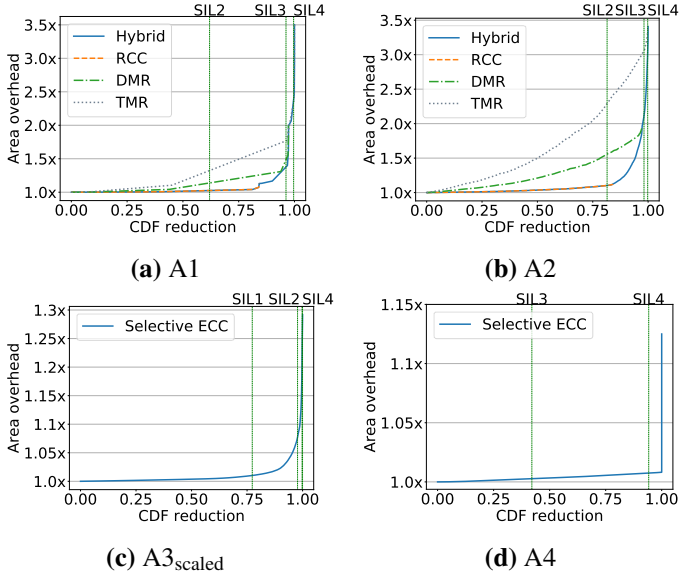
A3$_{\text{scaled}}$: Figure 5.14c shows the fraction of protected bits versus the CDF

rate reduction for different selection criteria in $A3_{scaled}$. We compare CEF with two heuristics: uniform and access-frequency-based selection. Access-frequency-based selection has been proposed for embedded applications [131, 163]. We find that the CEF-aware selection of bits results in an average 1.07× and 1.90× lower CDF rate for the same amount of protected bits compared to CS_volume and access-frequency. This is because the access-frequency-based heuristic captures the failure probability of structures within a single motion, but not the relative failure probabilities across different motions. Further, CEF-aware selection achieves 18.89× and 0.79× reduction in CDF rate than uniform and ideal error mitigation, respectively.

A4: Figure 5.14d compares different selection criteria for A4. We find that the CEF-aware selection of bits results in an average 1.02× reduction in CDF rate for the same amount of protected bits compared to CS_volume. In the proposed accelerator, the access-frequency for all the bits of a flattened octree is the same, and hence uniform and access-frequency-based selection given the same reduction in the CDF rate. Further, CEF-aware selection achieves 9.41× and 0.83× reduction in CDF rate than uniform/access-frequency and ideal error mitigation respectively.

Area savings using CEF-aware error mitigation

Further, we measure the area/power overheads to achieve different SILs using the CEF-aware error mitigation approach. In A1 and A2, on-chip storage elements consist of 15-bit registers that are accessed independently. We assume the use of latch hardening technique summarized in Table 5.4 in Sullivan et al. [240] to measure the overheads for A1 and A2. . Latch hardening is more suitable than ECC for error mitigation in registers that are accessed in parallel. In contrast, we use ECC (SEC-DED code) for $A3_{scaled}$ and A4 as these accelerators use SRAM for on-chip storage or DRAM (Table 5.1). For $A3_{scaled}$, each entry in SRAM consists of 24 bits. We assume an overhead of 7 bits (30%) to store the ECC bits. For A4, each entry in DRAM consists of 64 bits, with 8 bits overhead for ECC bits. We ignore the area overhead of the error detection/correction circuits themselves. Note that other approaches for ECC [86] can also be combined with CEF-aware selection. Figure 5.15 shows the storage area overhead versus CDF rate reduction



(a) A1 **(b)** A2

(c) $A3_{scaled}$ **(d)** A4

Figure 5.15: Area overhead versus CDF rate reduction for selective error mitigation in A1, A2, $A3_{scaled}$, and A4. (a)-(d) compare the area overhead for different CDF reduction values for all CDMs. Table 5.5 summarizes the area/power overhead for CEF-aware error mitigation for different safety levels and blanket protection of the entire on-chip memory.

for CEF-aware error mitigation for A1, A2, $A3_{scaled}$, and A4. SIL 1-SIL 4 markers are shown for CDF rate achievement for the Jaco2 robot and different safety standard levels. Table 5.5 compares the area/power overheads for different safety levels for all accelerators with the blanket protection of CDMs. CEF-aware selection results in significantly lower overhead for all levels.

Table 5.4: Area overhead and CDF rate reduction for different latch hardening techniques.

Latch type	Area Overhead	failure rate reduction
Strike Suppression (RCC [219])	1.15x	6.3x
Redundant Node (SEUT [219])	2x	37x
Triple Module Redundancy (TMR [160])	3.5x	1,000,000x

Table 5.5: Area/Power overhead for blanket and CEF-aware error mitigation for different SILs. In A1, A2, $A3_{scaled}$, and A4, on-chip storage elements consume $\sim 50\%/\sim 30\%$, $\sim 50\%/\sim 30\%$, $\sim 40\%/\sim 30\%$, and $\sim 98\%/\sim 98\%$ of the total area/power of the CDM, respectively.

Accel.	Blanket Error Mitigation	CEF-aware Error Mitigation			
		SIL1	SIL2	SIL3	SIL4
A1	125%/75%	-	1.2%/0.7%	18.8%/11.3%	68.9%/41.4%
A2	125%/75%	-	5.1%/3.1%	55.7%/33.4%	101.9%/61.1%
$A3_{scaled}$	12%/9%	0.4%/0.3%	3.1%/2.3%	8.8%/6.6%	11.4%/8.5%
A4	12%/12%	-	-	0.3%/0.3	0.7%/0.7%

CEF-aware error mitigation implementation

As mentioned in Section 5.2.2, CEF-aware selective error mitigation can be implemented by reliability-aware data placement in partially protected memory. We discuss possible implementations of selective error mitigation for CDMs studied in this work.

For A1 and A2, all on-chip registers in a CDC are accessed in parallel, and there are no address registers. In this case, each CDC contains a fixed number of protected registers. Selective error mitigation can be achieved by CEF-aware placement of data in CDC's protected and unprotected registers. For A3, each CDC consists of an SRAM. Instead of a single SRAM, protected and unprotected SRAMs with a fixed address range can be used in this case. Thus, selective error mitigation can be achieved by CEF-aware address assignment and placement, with minimal performance/energy overheads [165]. A4 uses a DRAM to store the data. In this case, ECC bits can be embedded into each data row with low overhead, as shown in [33]. Several papers have focused on a flexible partition of the register file and DRAM with low overheads incorporating selective error mitigation in systems using CPUs and GPUs [33, 168, 258, 280, 284]. The implementation and overheads of selective error-mitigation depend upon the accelerator's microarchitecture, with scope for co-designing microarchitecture and selective error-mitigation implementation.

5.6 Architectural Implications

In this section, we compare the reliabilities of the four CDMs to draw lessons for resilience-aware CDM design. Error mitigation can incur significant performance,

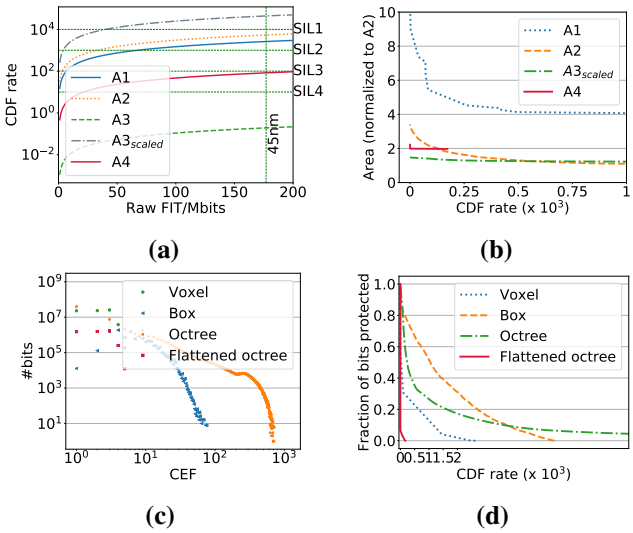


Figure 5.16: Comparison of different CDM architectures geometric representations. (a) compares the CDF rate for different Raw FIT values and technology nodes, (b) compares the CDM area versus CDF rate, (c) compares the CEF distribution, and (d) compares the fraction of protected bits versus CDF rate for different accelerators.

area, and energy overheads (e.g., 3.5× area and energy for TMR), depending upon the CDM architecture. Hence, we need to consider the overheads of error mitigation for making architectural decisions.

Figure 5.16a compares the CDF rates of accelerators A1, A2, A3, $A3_{scaled}$, and A4. Figure 5.16b compares the error mitigation overhead for these accelerators for different CDF rates. Even though $A3_{scaled}$ has the highest CDF rate (Figure 5.16a), ECC can be used for low overhead error mitigation in $A3_{scaled}$ as it uses SRAM.

Geometric representation of swept space: Figure 5.16c compares the CEF distribution of different geometric representations used in the CDMs studied for a motion set of the Jaco2 robot. We find that two aspects of the geometric representation mainly affect the CEF distribution and range: (1) redundancy, and the (2) volume covered by a structure. In the box-based representation, to take advantage of spatial locality, the optimization process converts the swept space into a set of boxes. Each box covers the maximum possible number of voxels in the swept space. This adds redundancy as some voxels are covered by multiple boxes, and

are hence not included in critical space. In the octree-based representation, nodes near the root node divide the space at a coarser level and represent a much larger volume. Also, there is no redundancy in the representation. These factors result in an overall higher CEF for octree. Conversely, in voxel- and flattened-octree-based representation, each structure/bit represents a single voxel, and so their CEF is less than 6 (maximum 6 surfaces of a voxel). Further, geometric representation determines the suitable storage structures and the error mitigation approach. For example, using SRAM/DRAM with ECC results in lower costs of error mitigation for stringent CDF rate requirements as shown in Figure 5.16d ($A3_{scaled}$ and A4).

Data reuse: Many accelerators use on-chip buffers to store frequently used data, and exploit data reuse to reduce memory accesses. As a result, a soft error-induced bit flip persists until it is overwritten by reloading the swept space data as explained in Section 5.3.3. Therefore, if the erroneous data is used for N collision queries, the CDF rate increases by $N/2$ times (Equation 5.2). However, frequent reloading of on-chip data increases DRAM accesses and incurs performance and energy overheads. Thus there is a trade-off between the overhead of error mitigation and DRAM accesses.

5.7 Related work

Fault Characterization: Several FI tools focus on CPUs and GPUs [65, 67, 97, 146, 148, 176, 218, 248, 249, 257]. TRIDENT [148] uses compiler information to estimate SDC probability without performing FI for CPU. gem5-Approxilyzer [257] uses gem5 simulator [19] for FI, and proposes a methodology for fault site pruning. GPU-Qin [65] proposed an FI methodology that balances representativeness and efficiency and performs FI on real GPU hardware. GPU FI tools [67, 176] use GPGPU-Sim [10] for FI, and propose fault site pruning for the GPU SIMT execution model. gpuFI-4 [218] proposes a detailed microarchitecture-level FI tool to characterize the cross-layer vulnerability for single and multiple-bit faults. SASSIFI [97] and NVBitFI [248] propose instrumentation-based FI tools for NVIDIA GPUs. These tools are useful for fault characterization of general purpose applications, but they cannot be easily applied to robotics accelerators that use specialized microarchitectures and instruction sets. More recent papers have studied the resilience of

DNN accelerators [104, 105, 147, 154, 205, 214, 250, 260], and autonomous vehicle systems [12, 109, 119–121, 262]. We focus, instead, on robotics accelerators processing spatial information. Due to the differences between the datapath and information processed by the DNN accelerators and motion planning accelerators, the fault propagation to the output is significantly different between them. Another body of work has proposed metrics to characterize the vulnerability of structures to faults [66, 170, 234, 235]. Mukherjee et al. [170] defined the Architectural Vulnerability Factor (AVF), and proposed Architectural Correct Execution analysis (ACE-analysis) to approximate AVF for microprocessor structures [170]. Sridharan et al. [234] proposed the Program Vulnerability Factor (PVF) to decouple the program's fault-masking effect from that of the microarchitecture, and quantify the vulnerability of a program. Fang et al. extended this work in ePVF [66] to consider only SDC-causing bits. Although useful for CPUs and GPUs, architectural or microarchitectural ACE-analysis is challenging to apply to accelerators due to differences in the ISA and workload. For MPAs in particular, whether a bit is ACE heavily depends upon the position of obstacles, and requires multiple simulations to estimate. The CEF gives a measure of the fraction of runtime for which a bit is ACE without such simulations.

Error Mitigation Techniques: There has been significant work on selective error mitigation techniques for CPUs [207], GPUs [161, 165, 184], and FPGAs [154, 193]. Mittal et al. [165] proposed compressing similar values in GPUs. Palframan et al. [184] analyzed GPGPU applications and proposed architectural modifications to reduce the magnitude of errors. Unfortunately, these methods are difficult to apply to MPAs due to differences in the ISA and microarchitecture. Reis et al. [207] and Mahmoud et al. [161] proposed software-level instruction replication to improve the resilience of CPUs and GPUs respectively. However, accelerators typically use complex custom instructions and perform more computations per instruction [41, 153, 171], and hence software-level duplication would result in significant overheads. Li et al. [148] examined the resilience properties of DNN accelerators and proposed a method to protect vulnerable bits. Guan et al. [89] proposed leveraging application-specific data properties in CNNs to minimize the error correction overhead. In contrast, we focus on accelerators processing spatial information.

Motion Planning: Many techniques have been proposed to accelerate motion

planning on CPUs and GPUs [8, 18, 76, 226]. However, these do not meet the energy and performance requirements of autonomous robots [153, 172]. ASIC-based and FPGA-based MPAs [11, 153, 172, 233, 278] focus on performance and energy optimizations, rather than resilience. Other work [43, 241] has studied resilient motion planning under sensor and communication faults, but not soft errors. Note that many of these techniques use the term *roadmap* instead of *motion set* to denote the same idea.

5.8 Conclusions and Future Work

Motion planning is a critical task in autonomous robots, and motion planning accelerators (MPAs) have been proposed to speed it up significantly. Collision detection is the most resource-consuming and safety-critical module in MPAs. In this work, we propose a spatially-aware reliability metric (CEF) for MPAs, based on the exposed surface area of critical space. We propose a CEF-aware mitigation strategy and Fault Injection (FI) method based on this metric. We also find that CEF-aware error mitigation achieves significant collision detection failure rate reduction, even while incurring low area and energy overheads. We find that CEF-aware FI results in $\sim 23,000\times$ speedup over exhaustive FI to identify the critical bits. Finally, we identify the architectural design parameters affecting the resilience and error mitigation overheads in MPAs.

There are several possible directions for future work. First, while we focus on single bit flips, both the reliability metric CEF, and the proposed error mitigation and FI methods can be extended to a multi-bit fault model. In particular, Phase 1 of FI, CEF measurement needs to be modified to include a multi-bit fault model, where CEF of a bit can be calculated as an average of CEF for different multi-bit fault combinations. Second, while we focused on probabilistic roadmap-based MPAs, the underlying ideas can be applied to other robotics accelerators that process spatial information. In this work, CEF is evaluated for CDMs that dedicate a significant area to storage structures for spatial data. Earlier work on FPGA-based accelerators [173] uses combinational logic to encode spatial data. The definition of CEF applies to erroneous combinational logic gates. Finally, our observations on the effect of the different design parameters on the resilience and error mitigation overhead open up

the direction of “resilience-aware” algorithm-hardware co-design.

Chapter 6

Conclusion and Future Work

This thesis explores different aspects of autonomous robots' computation pipeline. It focuses on energy-efficient acceleration of different computing tasks using algorithm and/or hardware optimization.

The first part of the thesis explored energy-efficient acceleration of robot motion planning. In this work, we propose a Spatially Aware Scheduler, a Cascaded Early-Exit Collision Detection Unit, and a Collision Prediction Unit. These three modules target different levels of parallelism and sources of redundant computation in motion planning. We further provide a motion planning hardware accelerator consisting of the proposed modules. MPAccel enables realtime motion planning with 3.5W power for a 7-DOF robotic arm. In this work, we focus on taking advantage of the physical spatial locality of objects and robot positions to reduce computation in motion planning. The proposed concept can be expanded for different motion planning algorithms and other computation tasks in autonomous robots, such as occupancy mapping.

The second part of the thesis focuses on improving deep regression networks and proposes a framework for regression by binary classification. In this work, we provide a taxonomy of different design aspects introduced in regression by binary classification formulation and conduct a systematic analysis of this design space. We explore the properties of suitable label encoding for regression and propose manually designed label encoding, as well as a label encoding learning approach. Our evaluation shows that label encoding can provide significant improvement in

regression error over direct regression, and provide an attractive generic approach suitable for a wide range of regression tasks. Further, label encoding has the potential to reduce inference computation by enabling the use of sparser and smaller models without significantly reducing accuracy. The main goal of this work is to establish this design space and demonstrate the potential of studying the design space. The future direction of this work includes a theoretical analysis of binary classification loss for regression tasks. Further, binary label encoding learning approaches can be explored to enable the use of binary cross entropy loss function. There is a huge scope for formally studying different design aspects introduced by the proposed framework.

The third part of the thesis focuses on improving the soft-error resilience of collision detection modules used in robotics. We propose an application-specific reliability metric, CEF. We further explore CEF-aware fault characterization and error mitigation. CEF is applied to a subset of hardware accelerators for robotics. However, the insight behind using exposed physical space to estimate failure probability can be applied to different hardware accelerators in robotics. CEF's adaption for other robotic accelerators is an interesting future direction.

Bibliography

[1] Design compiler. https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html, 2020.

[2] E. Ackerman. Jaco is a low-power robot arm that hooks to your wheelchair. https://spectrum.ieee.org/the-human-os/robotics/medical-robots/robot-arm-helps-disabled-11-year-old-girl-show-horse-in-competition, 2019.

[3] K. AG. Industrial robots from kuka, 2023. URL https://www.kuka.com/en-ca/products/robotics-systems/industrial-robots.

[4] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *J. Mach. Learn. Res.*, 1:113–141, Sept. 2001. doi:10.1162/15324430152733133.

[5] J. Amanatides and K. Choi. Ray tracing triangular meshes. 1995.

[6] N. Amato and L. Dale. Probabilistic roadmap methods are embarrassingly parallel. In *Proceedings 1999 IEEE International Conference on Robotics and Automation*, pages 688–694, 1999. doi:10.1109/ROBOT.1999.770055.

[7] AMD Xilinx. Device reliability report second half 2021 (ug116). https://docs.xilinx.com/r/en-US/ug116/Failure-Rate-Summary, 2022.

[8] N. Atay and B. Bayazit. A motion planning processor on reconfigurable hardware. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 125–132. IEEE, 2006. ISBN 0780395069. doi:10.1109/ROBOT.2006.1641172.

[9] F. Ayatolahi, B. Sangchoolie, R. Johansson, and J. Karlsson. A study of the impact of single bit-flip and double bit-flip errors on program execution. In *Proceedings of the International Conference on Computer Safety, Reliability,*

and Security, SAFECOMP, pages 265–276. Springer-Verlag, 2013. ISBN 9783642407925.

[10] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong, and T. M. Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174, 2009. doi:10.1109/ISPASS.2009.4919648.

[11] M. Bakhshalipour, S. B. Ehsani, M. Qadri, D. Guri, M. Likhachev, and P. B. Gibbons. Racod: Algorithm/hardware co-design for mobile robot path planning. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. Association for Computing Machinery, 2022. doi:10.1145/3470496.3527383.

[12] S. S. Banerjee, S. Jha, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer. Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data. DSN, pages 586–597. IEEE, 2018. ISBN 9781538655955. doi:10.1109/DSN.2018.00066.

[13] B. R. Bartoldson, A. S. Morcos, A. Barbu, and G. Erlebacher. The generalization-stability tradeoff in neural network pruning. *CoRR*, abs/1906.03728, 2019. URL http://arxiv.org/abs/1906.03728.

[14] A. Behera, Z. Wharton, P. Hewage, and S. Kumar. Rotation axis focused attention network (rafa-net) for estimating head pose. In *Computer Vision – ACCV 2020*, 2021.

[15] V. Belagiannis, C. Rupprecht, G. Carneiro, and N. Navab. Robust optimization for deep regression. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2830–2838, 2015. doi:10.1109/ICCV.2015.324.

[16] A. Berg, M. Oskarsson, and M. O'Connor. Deep ordinal regression with label diversity. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 2740–2747. IEEE, 2021.

[17] J. Bernhard, R. Gieselmann, K. Esterle, and A. Knoll. Experience-based heuristic search: Robust motion planning with deep q-learning. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3175–3182, 2018. doi:10.1109/ITSC.2018.8569436.

[18] J. Bialkowski, S. Karaman, and E. Frazzoli. Massively parallelizing the RRT and the RRT. In *International Conference on Intelligent Robots and Systems*,

IROS, pages 3513–3518, 2011. ISBN 9781612844541. doi:10.1109/IROS.2011.6048813.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, 2011. ISSN 0163-5964. doi:10.1145/2024716.2024718.

[20] G. E. Blelloch and B. M. Maggs. Parallel algorithms. *ACM Comput. Surv.*, 28(1):51–54, 1996. ISSN 0360-0300. doi:10.1145/234313.234339. URL https://doi.org/10.1145/234313.234339.

[21] R. Bohlin and L. Kavraki. Path planning using lazy prm. In *IEEE International Conference on Robotics and Automation*, pages 521–528, 2000. doi:10.1109/ROBOT.2000.844107.

[22] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to End Learning for Self-Driving Cars. *arXiv:1604.07316*, 2016.

[23] M. Bojarski, P. Yeres, A. Choromanaska, K. Choromanski, B. Firner, L. Jackel, and U. Muller. Explaining how a deep neural network trained with end-to-end learning steers a car. *arXiv:1704.07911*, 2017.

[24] B. Boroujerdian, R. Ghosal, J. Cruz, B. Plancher, and V. J. Reddi. Roborun: A robot runtime to exploit spatial heterogeneity. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 829–834, 2021. doi:10.1109/DAC18074.2021.9586280.

[25] R. Bose and S. Shrikhande. A note on a result in the theory of code construction. *Information and Control*, 2(2):183–194, 1959. doi:https://doi.org/10.1016/S0019-9958(59)90376-6.

[26] D. Bridge. Autonomous robot market to exhibit a remarkable growth of usd 11607.13 million by 2030. https://www.globenewswire.com/news-release/2023/02/02/2600592/0/en/Autonomous-Robot-Market-to-Exhibit-a-Remarkable-Growth-of-USD-11607-13-Million-by-2030-Size-Share-Trends-Key-Drivers-Growth-and-Opportunity-Analysis.html, 2023.

[27] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings of 27th International Symposium on Computer Architecture*, pages 83–94, 2000.

[28] A. Bulat and G. Tzimiropoulos. Human Pose Estimation via Convolutional Part Heatmap Regression. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Computer Vision – ECCV 2016*, pages 717–732, 2016.

[29] X. Burgos-Artizzu, P. Perona, and P. Dollár. Robust Face Landmark Estimation under Occlusion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1513–1520, 12 2013. doi:10.1109/ICCV.2013.191.

[30] Cadence Design Systems. Incisive functional safety simulator. https://www.cadence.com/en_US/home/tools/system-design-and-verification/simulation-and-testbench-verification/incisive-functional-safety-simulator.html, 2020.

[31] W. Cao, V. Mirjalili, and S. Raschka. Rank consistent ordinal regression for neural networks with application to age estimation. *Pattern Recognition Letters*, 140:325–331, 2020. doi:https://doi.org/10.1016/j.patrec.2020.11.008.

[32] Y. Cao, L. Chen, and Z. Zhang. Memory design for selective error protection. In *Proceedings of the IEEE International Conference on Computer Design*, ICCD, pages 133–140, 2015. doi:10.1109/ICCD.2015.7357094.

[33] Y. Cao, L. Chen, and Z. Zhang. Memory design for selective error protection. In *Proceedings of the IEEE International Conference on Computer Design*, ICCD, pages 133–140. IEEE Computer Society, 2015. ISBN 9781467371667. doi:10.1109/ICCD.2015.7357094.

[34] R. Cappel and C. Perry-Sullivan. Yield and cost challenges at 16nm and beyond. https://sst.semiconductor-digest.com/2016/02/yield-and-cost-challenges-at-16nm-and-beyond/, 2016.

[35] J. S. Cardoso and J. F. Pinto da Costa. Learning to Classify Ordinal Data: The Data Replication Method. *Journal of Machine Learning Research*, 8: 1393–1429, 2007.

[36] D. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 46–51 vol.2, May 1993. doi:10.1109/ROBOT.1993.292122.

[37] C.-K. Chang, S. Lym, N. Kelly, M. B. Sullivan, and M. Erez. Evaluating and accelerating high-fidelity error injection for hpc. In *Proceedings of the*

International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18. IEEE Press, 2018.

[38] J.-W. Chang, W. Wang, and M.-S. Kim. Efficient collision detection using a dual obb-sphere bounding volume hierarchy. *Computer-Aided Design*, 42: 50–57, 2010. doi:10.1016/j.cad.2009.04.010.

[39] A. Chatzidimitriou, P. Bodmann, G. Papadimitriou, D. Gizopoulos, and P. Rech. Demystifying soft error assessment strategies on arm cpus: Microarchitectural fault injection vs. neutron beam experiments. In *Proceedings of the International Conference on Dependable Systems and Networks*, DSN, pages 26–38, 2019.

[40] S. Chen. Driving-datasets. https://github.com/SullyChen/driving-datasets.

[41] Y.-H. Chen, J. Emer, and V. Sze. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016. doi:10.1109/ISCA.2016.40.

[42] R. Cheng, K. Shankar, and J. W. Burdick. Learning an optimal sampling distribution for efficient motion planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7485–7492, 2020. doi:10.1109/IROS45743.2020.9341245.

[43] A. L. Christensen, R. O'Grady, M. Birattari, and M. Dorigo. Fault detection in autonomous robots based on fault injection and learning. *Autonomous Robots*, pages 49–67, 2008. ISSN 09295593. doi:10.1007/s10514-007-9060-9.

[44] W. Chu and S. S. Keerthi. New approaches to support vector ordinal regression. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, page 145–152, 2005.

[45] M. Cissé, T. Artières, and P. Gallinari. Learning Compact Class Codes for Fast Inference in Large Multi Class Classification. In P. A. Flach, T. De Bie, and N. Cristianini, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 506–520. Springer Berlin Heidelberg, 2012.

[46] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2021.

[47] K. Crammer and Y. Singer. Pranking with Ranking. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, page 641–647. MIT Press, 2001.

[48] Daniel Sorin. Reliable robotics in academia and industry. Keynote 2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS), 2021.

[49] S. Darabi, M. Belbahri, M. Courbariaux, and V. P. Nia. Regularized binary network training. *arXiv preprint arXiv:1812.11800*, 2018.

[50] F. de Dinechin, M. Istoan, and G. Sergent. Fixed-point trigonometric functions on fpgas. *SIGARCH Comput. Archit. News*, 41(5):83–88, 2014. ISSN 0163-5964. doi:10.1145/2641361.2641375.

[51] Deloitte. Semiconductors - the next wave. https://www2.deloitte.com/content/dam/Deloitte/cn/Documents/technology-media-telecommunications/deloitte-cn-tmt-semiconductors-the-next-wave-en-190422.pdf, 2019.

[52] J. Denavit and R. S. Hartenberg. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. *Journal of Applied Mechanics*, 22(2): 215–221, 2021. ISSN 0021-8936. doi:10.1115/1.4011045.

[53] S. Designline. Fd soi benefits rise at 14nm. https://www.eetimes.com/fd-soi-benefits-rise-at-14nm/, 2016.

[54] T. G. Dietterich and G. Bakiri. Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995. doi:10.1613/jair.105.

[55] Diligent Robotics. Moxi helps clinical staff do more. https://diligentrobots.com/moxi, 2020.

[56] R. Díaz and A. Marathe. Soft labels for ordinal regression. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. doi:10.1109/CVPR.2019.00487.

[57] C. Ericson. *Real-Time Collision Detection*. CRC Press, Inc., 2004. ISBN 1558607323.

[58] C. Ericson. *Real-Time Collision Detection*. CRC Press, Inc., 2004. ISBN 1558607323.

[59] H. et al. 2.4 a distributed autonomous and collaborative multi-robot system featuring a low-power robot soc in 22nm cmos for integrated battery-powered minibots. In *2019 IEEE International Solid- State Circuits Conference*, 2019.

[60] J. et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 1–12, June 2017. doi:10.1145/3079856.3080246.

[61] I. Evron, E. Moroshko, and K. Crammer. Efficient Loss-Based Decoding on Graphs for Extreme Classification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 7233–7244, 2018.

[62] B. Eysenbach, R. Salakhutdinov, and S. Levine. *Search on the Replay Buffer: Bridging Planning and Reinforcement Learning*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[63] S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *ICML*, 2018.

[64] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Gool. Random forests for real time 3d face analysis. *International Journal of Computer Vision*, 101(3): 437–458, Feb. 2013.

[65] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi. Gpu-qin: A methodology for evaluating the error resilience of gpgpu applications. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 221–230, 2014. doi:10.1109/ISPASS.2014.6844486.

[66] B. Fang, Q. Lu, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi. epvf: An enhanced program vulnerability factor methodology for cross-layer resilience analysis. In *Proceedings of the International Conference on Dependable Systems and Networks*, DSN, pages 168–179, 2016.

[67] B. Fang, K. Pattabiraman, M. Ripeanu, and S. Gurumurthi. A systematic methodology for evaluating the error resilience of gpgpu applications. pages 3397–3411. IEEE Press, 2016.

[68] N. Farazmand, R. Ubal, and D. Kaeli. Statistical fault injection-based avf analysis of a gpu architecure. *IEEE Workshop on Silicon Errors in Logic*, 01 2012.

[69] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120, Brisbane, Australia, 2018.

[70] D. Ferguson, N. Kalra, and A. Stentz. Replanning with rrts. In *Proceedings 2006 IEEE International Conference on Robotics and Automation*, pages 1243–1248, 2006. doi:10.1109/ROBOT.2006.1641879.

[71] A. Fishman, A. Murali, C. Eppner, B. N. Peele, B. Boots, and D. Fox. Motion policy networks. In *Conference on Robot Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:253098016.

[72] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao. Deep Ordinal Regression Network for Monocular Depth Estimation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[73] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014. doi:10.1109/IROS.2014.6942976.

[74] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa. Batch informed trees (bit*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research*, 39(5):543–567, 2020. doi:10.1177/0278364919890396.

[75] B.-B. Gao, H.-Y. Zhou, J. Wu, and X. Geng. Age estimation using expectation of label distribution learning. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 712–718, 7 2018. doi:10.24963/ijcai.2018/99.

[76] R. Gayle, P. Segars, M. Lin, and D. Manocha. Path planning for deformable robots in complex environments. In *Robotics: Science and Systems*, pages 225–232, 2005. doi:10.15607/RSS.2005.I.030.

[77] R. Geraerts and M. Overmars. Creating high-quality paths for motion planning. *International Journal of Robotics Research*, 26, 08 2007. doi:10.1177/0278364907079280.

[78] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. *CoRR*, abs/2103.13630, 2021. URL https://arxiv.org/abs/2103.13630.

[79] D. Glas, T. Kanda, H. Ishiguro, and N. Hagita. Teleoperation of multiple social robots. *IEEE Transactions on Systems, Man, and Cybernetics - TSMC*, 42:530–544, 05 2012. doi:10.1109/TSMCA.2011.2164243.

[80] T. Globe and Mail. Turning autonomous cars into profit won't be easy. https://www.theglobeandmail.com/opinion/article-turning-autonomous-cars-into-profit-wont-be-easy/, 2019.

[81] D. A. G. Gonçalves de Oliveira, L. L. Pilla, T. Santini, and P. Rech. Evaluation and mitigation of radiation-induced soft errors in graphics processing units. *IEEE Transactions on Computers*, 65:791–804, 2016. doi:10.1109/TC.2015.2444855.

[82] Google. System architecture, 2020. URL https://cloud.google.com/tpu/docs/system-architecture-tpu-vm.

[83] Google. Edge tpu, 2023. URL https://cloud.google.com/edge-tpu.

[84] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH, pages 171–180. Association for Computing Machinery, 1996. ISBN 0897917464.

[85] S. Gottschalk, M. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30, 10 1997. doi:10.1145/237170.237244.

[86] J. Gracia-Moran, L. Saiz-Adalid, J. Baraza-Calvo, and P. Gil. Correction of adjacent errors with low redundant matrix error correction codes. In *2018 Eighth Latin-American Symposium on Dependable Computing (LADC)*, 2018. doi:10.1109/LADC.2018.00021.

[87] F. Gray. Pulse code communication, 1953.

[88] J. Groopman. Robots that care. https://www.newyorker.com/magazine/2009/11/02/robots-that-care, 2009.

[89] H. Guan, L. Ning, Z. Lin, X. Shen, H. Zhou, and S.-H. Lim. In-Place Zero-Space Memory Protection for CNN. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances*

in Neural Information Processing Systems, pages 5734–5743. Curran Associates, Inc., 2019.

[90] F. K. Gustafsson, M. Danelljan, G. Bhat, and T. B. Schön. Energy-based models for deep probabilistic regression. In *Computer Vision – ECCV 2020*, pages 325–343. Springer International Publishing, 2020.

[91] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar. OpenRAM: An open-source memory compiler. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6, Nov. 2016. doi:10.1145/2966986.2980098. ISSN: 1558-2434.

[92] M. Hamandi, M. D'Arcy, and P. Fazli. Deepmotion: Learning to navigate like humans. *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, pages 1–7, 2018. URL https://api.semanticscholar.org/CorpusID:4048147.

[93] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, volume 16, pages 243–254, 2016. doi:10.1109/ISCA.2016.30.

[94] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1510.00149.

[95] Y. Han, Y. Yang, X. Chen, and S. Lian. Dadu series - fast and efficient robot accelerators. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–8, 2020.

[96] I. S. Haque and V. S. Pande. Hard data on soft errors: A large-scale assessment of real-world error rates in gpgpu. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 691–696, 2010. doi:10.1109/CCGRID.2010.84.

[97] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer. SASSIFI: An architecture-level fault injection tool for GPU application resilience evaluation. In *IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS, pages 249–258, 2017. ISBN 9781538638897. doi:10.1109/ISPASS.2017.7975296.

[98] S. K. S. Hari, M. Sullivan, T. Tsai, and S. W. Keckler. Making convolutions resilient via algorithm-based error detection techniques. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2021. doi:10.1109/TDSC.2021.3063083.

[99] K. Hauser. Lazy collision checking in asymptotically-optimal motion planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, ICRA, pages 2951–2957, 2015. doi:10.1109/ICRA.2015.7139603.

[100] K. Hauser. *Robust Contact Generation for Robot Simulation with Unstructured Meshes*, pages 357–373. Springer, Cham, 2016. ISBN 978-3-319-28870-3. doi:10.1007/978-3-319-28872-7_21.

[101] K. Hauser and V. Ng-Thow-Hing. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. In *2010 IEEE International Conference on Robotics and Automation*, pages 2493–2498, 2010. doi:10.1109/ROBOT.2010.5509683.

[102] K. K. Hauser and J. Latombe. Integrating task and prm motion planning : Dealing with many infeasible motion planning queries. 2009.

[103] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[104] Y. He, P. Balaprakash, and Y. Li. Fidelity: Efficient resilience analysis framework for deep learning accelerators. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 270–281, 2020. doi:10.1109/MICRO50266.2020.00033.

[105] Y. He, M. Hutton, S. Chan, R. De Gruijl, R. Govindaraju, N. Patil, and Y. Li. Understanding and mitigating hardware failures in deep learning training systems. In *ISCA*, 2023.

[106] J. L. Hennessy and D. A. Patterson. *Computer Architecture, a quantitative approach*. Morgan Kaufmann, 2012. ISBN 9780123838728.

[107] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34, 04 2013. doi:10.1007/s10514-012-9321-0.

[108] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

[109] Y.-S. Hsiao, Z. Wan, T. Jia, R. Ghosal, A. Mahmoud, A. Raychowdhury, D. Brooks, G.-Y. Wei, and V. J. Reddi. Mavfi: An end-to-end fault analysis framework with anomaly detection and recovery for micro aerial vehicles. In *DATE*, 2023.

[110] H.-W. Hsu, T.-Y. Wu, S. Wan, W. H. Wong, and C.-Y. Lee. Quatnet: Quaternion-based head pose estimation with multiregression loss. *IEEE Transactions on Multimedia*, 21(4):1035–1046, 2019. doi:10.1109/TMM.2018.2866770.

[111] C.-M. Huang and S.-H. Hsu. Efficient path planning for a microrobot passing through environments with narrow passages. *Micromachines*, 13: 1935, 11 2022. doi:10.3390/mi13111935.

[112] J. Huh and D. D. Lee. Efficient sampling with q-learning to guide rapidly exploring random trees. *IEEE Robotics and Automation Letters*, 3(4): 3868–3875, 2018. doi:10.1109/LRA.2018.2856927.

[113] B. Ichter, J. Harrison, and M. Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094, 2018. doi:10.1109/ICRA.2018.8460730.

[114] B. Ichter, E. Schmerling, T.-W. E. Lee, and A. Faust. Learned critical probabilistic roadmaps for robotic motion planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9535–9541, 2020. doi:10.1109/ICRA40945.2020.9197106.

[115] IEC. Functional safety and iec 61508. http://www.iec.ch/functionalsafety/, 2020.

[116] F. B. Insights. The global autonomous mobile robots market... https://www.fortunebusinessinsights.com/autonomous-mobile-robots-market-105055, 2022.

[117] C. L. Jackins and S. L. Tanimoto. Oct-trees and their use in representing three-dimensional objects. *Computer Graphics and Image Processing*, pages 249–270, 1980. ISSN 0146-664X. doi:https://doi.org/10.1016/0146-664X(80)90055-6.

[118] L. Jaillet and T. Siméon. A prm-based motion planner for dynamically changing environments. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2:1606–1611 vol.2, 2004.

[119] S. Jha, S. S. Banerjee, J. Cyriac, Z. T. Kalbarczyk, and R. K. Iyer. AVFI: Fault Injection for Autonomous Vehicles. DSN-W, pages 55–56. IEEE, 2018. ISBN 9781538655955. doi:10.1109/DSN-W.2018.00027.

[120] S. Jha, S. Banerjee, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer. ML-Based Fault Injection for Autonomous Vehicles: A Case for Bayesian Fault Injection. In *Proceedings of the International Conference on Dependable Systems and Networks*, DSN, pages 112–124, 2019. ISBN 9781728100562. doi:10.1109/dsn.2019.00025.

[121] S. Jha, S. Cui, T. Tsai, S. K. S. Hari, M. B. Sullivan, Z. T. Kalbarczyk, S. W. Keckler, and R. K. Iyer. Exploiting temporal data diversity for detecting safety-critical faults in av compute systems. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2022.

[122] T. Jia, E.-Y. Yang, Y.-S. Hsiao, J. Cruz, D. Brooks, G.-Y. Wei, and V. J. Reddi. Omu: A probabilistic 3d occupancy mapping accelerator for real-time octomap at the edge. In *Proceedings of the 2022 Conference Exhibition on Design Automation Test in Europe*, page 909–914, 2022. ISBN 9783981926361.

[123] L. Jin, X. Shu, K. Li, Z. Li, G.-J. Qi, and J. Tang. Deep ordinal hashing with spatial attention. *Transaction on Image Processing*, 28(5):2173–2186, 2019.

[124] H. T. K., A. Balachandran, and S. Shah. Optimal whole-body motion planning of humanoids in cluttered environments. *Robotics and Autonomous Systems*, 118:263–277, 2019. ISSN 0921-8890. doi:https://doi.org/10.1016/j.robot.2019.04.004. URL https://www.sciencedirect.com/science/article/pii/S0921889017307108.

[125] M. Kaliorakis, D. Gizopoulos, R. Canal, and A. Gonzalez. Merlin: Exploiting dynamic instruction behavior for fast and accurate microarchitecture level reliability assessment. In *ISCA*, 2017.

[126] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *International Journal of Robotic Research - IJRR*, 30:846–894, 06 2011. doi:10.1177/0278364911406761.

[127] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12, 1996. doi:10.1109/70.508439.

[128] L. E. Kavraki and S. M. LaValle. *Motion Planning*. Springer Berlin Heidelberg, 2008.

[129] T. Kevan. Will robots find a place in the smart home? https://www.digitalengineering247.com/article/will-robots-find-a-place-in-the-smart-home, 2019.

[130] J. T. Kider, M. Henderson, M. Likhachev, and A. Safonova. High-dimensional planning on the gpu. In *2010 IEEE International Conference on Robotics and Automation*, pages 2515–2522, May 2010. doi:10.1109/ROBOT.2010.5509470.

[131] S. Kim and A. Somani. Area efficient architectures for information integrity in cache memories. In *Proceedings of the 26th International Symposium on Computer Architecture*, pages 246–255, 1999. doi:10.1109/ISCA.1999.765955.

[132] KINOVA. Kinova jaco assistive robot. https://www.kinovarobotics.com/sites/default/files/KINO-2018-Bro-Assistive-ZH_YUL-06-R-Web.pdf, 2018.

[133] J. Klosowski, M. Held, J. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, 1998. doi:10.1109/2945.675649.

[134] A. Korchut, S. Szklener, C. Abdelnour, N. Tantinya, J. Hernandez-Farigola, J. Ribes, U. Skrobas, K. Grabowska-Aleksandrowicz, D. Szczesniak-Stanczyk, and K. Rejdak. Challenges for service robots—requirements of elderly adults with cognitive impairments. *Frontiers in Neurology*, 8, 06 2017. doi:10.3389/fneur.2017.00228.

[135] S. Krishnan, Z. Wan, K. Bhardwaj, P. Whatmough, A. Faust, S. Neuman, G.-Y. Wei, D. Brooks, and V. J. Reddi. Automatic domain-specific soc design for autonomous unmanned aerial vehicles. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 300–317, 2022. doi:10.1109/MICRO56248.2022.00033.

[136] A. Kumar, T. K. Marks, W. Mou, Y. Wang, M. Jones, A. Cherian, T. Koike-Akino, X. Liu, and C. Feng. LUVLi face alignment: Estimating Landmarks' location, uncertainty, and visibility likelihood. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2020. doi:10.1109/CVPR42600.2020.00826.

[137] T. Kurutach, A. Tamar, G. Yang, S. Russell, and P. Abbeel. Learning plannable representations with causal infogan. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8747–8758, Red Hook, NY, USA, 2018. Curran Associates Inc.

[138] M. Köstinger, P. Wohlhart, P. M. Roth, and H. Bischof. Annotated Facial Landmarks in the Wild: A large-scale, real-world database for facial landmark localization. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 2144–2151, 2011.

[139] S. LaValle and J. Kuffner. Randomized kinodynamic planning. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, pages 473–479 vol.1, 1999. doi:10.1109/ROBOT.1999.770022.

[140] S. M. LaValle. *Planning Algorithms*. 2006. ISBN 9780521862059. URL http://lavalle.pl/planning/.

[141] S. M. LaValle. Motion planning. *IEEE Robotics Automation Magazine*, pages 79–89, 2011.

[142] L. Lee, E. Parisotto, D. S. Chaplot, E. Xing, and R. Salakhutdinov. Gated path planning networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2947–2955, 2018.

[143] J. Lengyel, M. Reichert, B. R. Donald, and D. P. Greenberg. Real-time robot motion planning using rasterizing computer graphics hardware. *SIGGRAPH Comput. Graph.*, 24(4):327–335, 1990. ISSN 0097-8930. doi:10.1145/97880.97915.

[144] P. Leven and S. Hutchinson. A framework for real-time path planning in changing environments. *The International Journal of Robotics Research*, pages 999–1030, 2002. doi:10.1177/027836402021012001.

[145] R. Leveugle, A. Calvez, P. Maistri, and P. Vanhauwaert. Statistical fault injection: Quantified error and confidence. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE, pages 502–506, 2009. ISBN 9783981080155.

[146] G. Li, K. Pattabiraman, C.-Y. Cher, and P. Bose. Understanding error propagation in gpgpu applications. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 240–251, 2016. doi:10.1109/SC.2016.20.

[147] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler. Understanding error propagation in deep learning neural network (DNN) accelerators and applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC, pages 1–12, 2017.

[148] G. Li, K. Pattabiraman, S. K. S. Hari, M. Sullivan, and T. Tsai. Modeling Soft-Error propagation in programs. DSN, pages 27–38. IEEE, 2018. ISBN 9781538655955. doi:10.1109/DSN.2018.00016.

[149] L. Li and H.-T. Lin. Ordinal regression by extended binary classification. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pages 865–872, 2006.

[150] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach. Learn. Res.*, 18(1):6765–6816, jan 2017. ISSN 1532-4435.

[151] S. Li and N. T. Dantam. Sample-driven connectivity learning for motion planning in narrow passages. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5681–5687, 2023. doi:10.1109/ICRA48891.2023.10161339.

[152] W. Li, X. Huang, J. Lu, J. Feng, and J. Zhou. Learning probabilistic ordinal embeddings for uncertainty-aware regression. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13891–13900. IEEE Computer Society, jun 2021. doi:10.1109/CVPR46437.2021.01368.

[153] S. Lian, Y. Han, X. Chen, Y. Wang, and H. Xiao. Dadu-p: A scalable accelerator for robot motion planning in a dynamic environment. In *Proceedings of the Annual Design Automation Conference*, DAC. Association for Computing Machinery, 2018. ISBN 9781450357005.

[154] F. Libano, B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech. Selective hardening for neural networks in fpgas. *IEEE Transactions on Nuclear Science*, pages 216–222, 2019. doi:10.1109/TNS.2018.2884460.

[155] W. Libaw and L. Craig. A photoelectric decimal-coded shaft digitizer. *Electronic Computers, Transactions of the I.R.E. Professional Group on*, EC-2:1–4, 10 1953.

[156] H. Liu, R. Wang, S. Shan, and X. Chen. Deep supervised hashing for fast image retrieval. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[157] J. Liu, N. Hegde, and M. Kulkarni. Hybrid cpu-gpu scheduling and execution of tree traversals. In *Proceedings of the 2016 International Conference on Supercomputing*. Association for Computing Machinery, 2016. ISBN 9781450343619. doi:10.1145/2925426.2926261.

[158] X. Luo, H. Wang, D. Wu, C. Chen, M. Deng, J. Huang, and X.-S. Hua. A survey on deep hashing methods. *ACM Trans. Knowl. Discov. Data*, 2022. ISSN 1556-4681. doi:10.1145/3532624.

[159] Lynxmotion. Al5d robot arm specs. http://www.lynxmotion.com/driver.aspx?Topic=specs04, 2020.

[160] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, 6 (2):200–209, 1962.

[161] A. Mahmoud, S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler. Optimizing software-directed instruction replication for GPU error detection. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC, pages 1–12, Dallas, Texas, 2018.

[162] T. Meany. Functional Safety for Integrated Circuits. Technical Article, Analog Devices https://www.analog.com/media/en/technical-documentation/tech-articles/a54121%20functional-safety-for-integrated-circuits.pdf, 2012.

[163] M. Mehrara and T. Austin. Exploiting selective placement for low-cost memory protection. *Transactions on Architecture and Code Optimization*, pages 1–24, 2008. ISSN 15443566. doi:10.1145/1455650.1455653.

[164] S. Mitra, T. Karnik, N. Seifert, and M. Zhang. Logic soft errors in sub-65nm technologies design and CAD challenges. In *Proceedings of the Annual Design Automation Conference*, pages 2–4, 2005. doi:10.1145/1065579.1065585. ISSN: 0738-100X.

[165] S. Mittal, H. Wang, A. Jog, and J. S. Vetter. Design and Analysis of Soft-Error Resilience Mechanisms for GPU Register File. In *2017 30th International Conference on VLSI Design (VLSID)*, pages 409–414, 2017. doi:10.1109/VLSID.2017.14.

[166] T. Mitzner, T. Chen, C. Kemp, and W. Rogers. Identifying the potential for robotics to assist older adults in different living environments. *International journal of social robotics*, 6:213–227, 04 2014. doi:10.1007/s12369-013-0218-7.

[167] D. Molina, K. Kumar, and S. Srivastava. Learn and link: Learning critical regions for efficient planning. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10605–10611, 2020. doi:10.1109/ICRA40945.2020.9196833.

[168] P. Montesinos, W. Liu, and J. Torrellas. Using register lifetime predictions to protect register files against soft errors. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN, 2007. doi:10.1109/DSN.2007.99.

[169] H. P. Moravec. Puma. https://www.britannica.com/technology/PUMA-robot, 2021.

[170] S. S. Mukherjee, C. Weaver, J. Emer, S. K. Reinhardt, and T. Austin. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. In *Proceedings of the ACM/IEEE International Symposium on Microarchitecture*, MICRO, pages 29–40. IEEE Computer Society, 2003. ISBN 076952043X.

[171] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin. The microarchitecture of a real-time robot motion planning accelerator. In *Proceedings of the ACM/IEEE International Symposium on Microarchitecture*, MICRO. IEEE Press, 2016. ISBN 9781509035083. doi:10.1109/MICRO.2016.7783748.

[172] S. Murray, W. Floyd-Jones, Y. Qi, D. J. Sorin, and G. Konidaris. Robot motion planning on a chip. In *Robotics: Science and Systems*, 2016. doi:10.15607/rss.2016.xii.004.

[173] S. Murray, W. Floyd-jones, G. Konidaris, and D. J. Sorin. A Programmable Architecture for Robot Motion Planning Acceleration. In *International Conference on Application-specific Systems, Architectures and Processors*, ASAP, pages 185–188. IEEE, 2019.

[174] N. Nethercote and J. Seward. Valgrind: A framework for heavyweight dynamic binary instrumentation. *SIGPLAN Not.*, 42(6):89–100, jun 2007. ISSN 0362-1340. doi:10.1145/1273442.1250746. URL https://doi.org/10.1145/1273442.1250746.

[175] R. Neugebauer, G. Antichi, J. F. Zazo, Y. Audzevich, S. López-Buedo, and A. W. Moore. Understanding pcie performance for end host networking. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 327–341. Association for Computing Machinery, 2018. ISBN 9781450355674. doi:10.1145/3230543.3230560.

[176] B. Nie, L. Yang, A. Jog, and E. Smirni. Fault site pruning for practical reliability analysis of GPGPU applications. In *Proceedings of the ACM/IEEE International Symposium on Microarchitecture (MICRO)*, 2018. ISBN 978-1-5386-6240-3. doi:10.1109/MICRO.2018.00066.

[177] Z. Niu, M. Zhou, L. Wang, X. Gao, and G. Hua. Ordinal regression with multiple output CNN for age estimation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[178] NVIDIA. Nvidia a100 tensor core gpu, 2022. URL https://www.nvidia.com/en-us/data-center/a100/.

[179] NVIDIA. Jetson modules, 2023. URL https://developer.nvidia.com/embedded/jetson-modules.

[180] Nvprof. command line profiling tool. http://docs.nvidia.com/cuda/profiler-users-guide/, 2023.

[181] OHMNILABS. The importance of low-cost robotics. https://ohmnilabs.com/content/the-importance-of-low-cost-robotics/, 2019.

[182] P. S. Ostler, M. P. Caffrey, D. S. Gibelyou, P. S. Graham, K. S. Morgan, B. H. Pratt, H. M. Quinn, and M. J. Wirthlin. Sram fpga reliability analysis for harsh radiation environments. *IEEE Transactions on Nuclear Science*, 2009. doi:10.1109/TNS.2009.2033381.

[183] T. Pachidis, C. Sgouros, V. G. Kaburlasos, E. Vrochidou, T. Kalampokas, K. Tziridis, A. Nikolaou, and G. A. Papakostas. Forward kinematic analysis of jaco2 robotic arm towards implementing a grapes harvesting robot. In *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2020. doi:10.23919/SoftCOM50211.2020.9238297.

[184] D. J. Palframan, N. S. Kim, and M. H. Lipasti. Precision-aware soft error protection for GPUs. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, pages 49–59, 2014. doi:10.1109/HPCA.2014.6835966.

[185] H. Pan, H. Han, S. Shan, and X. Chen. Mean-variance loss for deep age estimation from a face. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5285–5294, 2018. doi:10.1109/CVPR.2018.00554.

[186] J. Pan and D. Manocha. Gpu-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research*, 31(2): 187–200, 2012. doi:10.1177/0278364911429335.

[187] J. Pan and D. Manocha. Gpu-based parallel collision detection for fast motion planning. *The International Journal of Robotics Research*, 31(2): 187–200, 2012. doi:10.1177/0278364911429335.

[188] J. Pan, C. Lauterbach, and D. Manocha. g-planner: Real-time motion planning and global navigation using gpus. In *Proceedings of the National Conference on Artificial Intelligence*, volume 2, 01 2010.

[189] G. Papadimitriou and D. Gizopoulos. Demystifying the system vulnerability stack: Transient fault effects across the layers. In *48th Annual International Symposium on Computer Architecture (ISCA)*, 2021.

[190] G. Papadimitriou and D. Gizopoulos. Avgi: Microarchitecture-driven, fast and accurate vulnerability assessment. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 935–948, 2023. doi:10.1109/HPCA56546.2023.10071105.

[191] K. Pingali, M. Kulkarni, D. Nguyen, M. Burtscher, M. Mendez-Lojo, D. Prountzos, X. Sui, and Z. Zhong. Amorphous data-parallelism in irregular algorithms. *regular tech report TR-09-05, The University of Texas at Austin*, 2009.

[192] E. Plaku and L. Kavraki. Distributed sampling-based roadmap of trees for large-scale motion planning. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 3868–3873, April 2005. doi:10.1109/ROBOT.2005.1570711.

[193] B. Pratt, M. Caffrey, P. Graham, K. Morgan, and M. Wirthlin. Improving fpga design robustness with partial tmr. In *2006 IEEE International Reliability Physics Symposium Proceedings*, pages 226–232, 2006. doi:10.1109/RELPHY.2006.251221.

[194] S. Prokudin, P. Gehler, and S. Nowozin. Deep directional statistics: Pose estimation with uncertainty quantification. In *Computer Vision – ECCV 2018*, pages 542–559. Springer International Publishing, 2018. ISBN 978-3-030-01240-3.

[195] N. Pérez-Higueras, F. Caballero, and L. Merino. Learning human-aware path planning with fully convolutional networks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5897–5902, 2018. doi:10.1109/ICRA.2018.8460851.

[196] L. Qian, C. Wang, W. Liu, F. Lombardi, and J. Han. Design and evaluation of an approximate wallace-booth multiplier. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1974–1977, 2016. doi:10.1109/ISCAS.2016.7538962.

[197] A. Qureshi, Y. Miao, A. Simeonov, and M. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, PP:1–19, 08 2020. doi:10.1109/TRO.2020.3006716.

[198] A. Qureshi, Y. Miao, A. Simeonov, and M. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, PP:1–19, 08 2020. doi:10.1109/TRO.2020.3006716.

[199] A. H. Qureshi and M. C. Yip. Deeply informed neural sampling for robot motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6582–6588. IEEE, 2018.

[200] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE, 2019.

[201] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 37(1):48–66, 2021. doi:10.1109/TRO.2020.3006716.

[202] A. H. Qureshi, Y. Miao, A. Simeonov, and M. C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *IEEE Transactions on Robotics*, 37:48–66, 2021.

[203] J. Ramsey. 40 https://www.autoblog.com/2020/05/11/car-electronics-cost-semiconductor-chips/, 2020.

[204] S. Raschka. MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack. *Journal of Open Source Software*, 3(24):638, Apr. 2018. doi:10.21105/joss.00638.

[205] B. Reagen, U. Gupta, L. Pentecost, P. Whatmough, S. K. Lee, N. Mulholland, D. Brooks, and G.-Y. Wei. Ares: A framework for quantifying the resilience of deep neural networks. In *DAC*, 2018. ISBN 9781450357005.

[206] Realtime Robotics, Inc. Realtime robotics. https://rtr.ai/, 2020.

[207] G. Reis, J. Chang, N. Vachharajani, R. Rangan, and D. August. SWIFT: software implemented fault tolerance. In *International Symposium on Code Generation and Optimization*, pages 243–254, 2005. doi:10.1109/CGO.2005.34.

[208] Rethink Robotics. Baxter, 2013. URL https://robots.ieee.org/robots/baxter/.

[209] R. B. Review. Kinova honored for jaco robotic arm. https://www.roboticsbusinessreview.com/health-medical/kinova_honored_for_jaco_robotic_arm/, 2014.

[210] K. Ricanek and T. Tesafaye. Morph: a longitudinal image database of normal adult age-progression. In *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*, pages 341–345, 2006. doi:10.1109/FGR.2006.78.

[211] M. Riera, A. Canal, Ramon anf Gonzalez, J. Abella, M. Anglada, and M. Torrents. Soft-error vulnerability evolution:a 4d study (bulk/soi, planar/finfet). In *International Workshop on Reliability and Aging in Forthcoming Electronic Systems*, 2015.

[212] M. Rouman. Siemens and realtime robotics accelerate the integration of industrial robotic workcells. https://blogs.sw.siemens.com/tecnomatix/siemens-and-realtime-robotics-accelerate-the-integration-of-industrial-robotic-workcells/, 2020.

[213] N. Ruiz, E. Chong, and J. M. Rehg. Fine-grained head pose estimation without keypoints. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[214] A. Ruospo, G. Gavarini, C. de Sio, J. Guerrero, L. Sterpone, M. S. Reorda, E. Sanchez, R. Mariani, J. Aribido, and J. Athavale. Assessing convolutional neural networks reliability through statistical fault injections. In *DATE*, 2023.

[215] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2015.

[216] C. Sagonas, G. Tzimiropoulos, S. Zafeiriou, and M. Pantic. 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 397–403, 2013.

[217] O. Salzman. Sampling-based robot motion planning. *Communications of the ACM*, pages 54–63, 2019. ISSN 0001-0782. doi:10.1145/3318164. URL https://doi.org/10.1145/3318164.

[218] D. Sartzetakis, G. Papadimitriou, and D. Gizopoulos. gpufi-4: A microarchitecture-level framework for assessing the cross-layer resilience of nvidia gpus. In *IEEE ISPASS*, 2022.

[219] N. Seifert, V. Ambrose, B. Gill, Q. Shi, R. Allmon, C. Recchia, S. Mukherjee, N. Nassif, J. Krause, J. Pickholtz, and A. Balasubramanian. On the radiation-induced soft error performance of hardened sequential elements in advanced bulk cmos technologies. In *IEEE International Reliability Physics Symposium*, pages 188–197, 2010. doi:10.1109/IRPS.2010.5488831.

[220] J. H. Seo, H. Lee, and K.-D. Kim. A parallelization algorithm for real-time path shortening of high-dofs manipulator. *IEEE Access*, 9:123727–123741, 2021. doi:10.1109/ACCESS.2021.3109744.

[221] D. Shah and T. M. Aamodt. Learning label encodings for deep regression. In *International Conference on Learning Representations*, May 2023. URL https://openreview.net/pdf?id=k60XE_b0Ix6.

[222] D. Shah and T. M. Aamodt. Collision prediction for robotics accelerators. In *Proceedings of the 51st Annual International Symposium on Computer Architecture [jsut accepted]*, ISCA'24. Association for Computing Machinery, 2023.

[223] D. Shah, Z. Y. Xue, and T. M. Aamodt. Label encoding for regression networks. In *International Conference on Learning Representations*, April 2022. URL https://openreview.net/pdf?id=8WawVDdKqlL.

[224] D. Shah, Z. Y. Xue, K. Pattabiraman, and T. M. Aamodt. Characterizing and improving resilience of accelerators to memory errors in autonomous robots. *ACM Transactions on Cyber-Physical Systems*, oct 2023. ISSN 2378-962X. doi:10.1145/3627828. URL https://doi.org/10.1145/3627828. Just Accepted.

[225] D. Shah, N. Yang, and T. M. Aamodt. Energy-efficient realtime motion planning. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700958. doi:10.1145/3579371.3589092. URL https://doi.org/10.1145/3579371.3589092.

[226] X. Shi, L. Cao, D. Wang, L. Liu, G. You, S. Liu, and C. Wang. HERO: Accelerating Autonomous Robotic Tasks with FPGA. IROS, pages 7766–7772, 2018. ISBN 9781538680940. doi:10.1109/IROS.2018.8593522.

[227] Z. Shiller and S. Sharma. High speed on-line motion planning in cluttered environments. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 596–601, 2012. doi:10.1109/IROS.2012.6385475.

[228] J. Shun. *Shared-Memory Parallelism Can Be Simple, Fast, and Scalable*, volume 15. Association for Computing Machinery and Morgan, 2017. ISBN 9781970001914.

[229] Y. Song, Q. Kang, and W. P. Tay. Error-Correcting Output Codes with Ensemble Diversity for Robust Learning in Neural Networks. *AAAI*, 2021.

[230] Z. Song, R. Wang, D. Ru, Z. Peng, H. Huang, H. Zhao, X. Liang, and L. Jiang. Approximate random dropout for dnn training acceleration in gpgpu. In *2019 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 108–113, 2019. doi:10.23919/DATE.2019.8715135.

[231] D. Sorin, W. Floyd-Jones, S. Murray, G. Konidaris, and W. Walker. Apparatus, method and article to facilitate motion planning of an autonomous vehicle in an environment having dynamic objects , 2019. Patent No. US11292456B2, Filed April 1, 2019, Issued April 5, 2022.

[232] D. J. Sorin and G. Konidaris. Enabling faster, more capable robots with real-time motion planning, 2018. URL https://spectrum.ieee.org/enabling-faster-more-capable-robots-with-real-time-motion-planning.

[233] D. J. Sorin, G. Konidaris, W. Floyd-Jones, and S. Murray. Motion Planning for Aotonomous Vehicles and Reconfigurable Motion Planning Processor, 2019. Patent No. US20190163191A1, Filed June 9, 2017, Issued May 30, 2019.

[234] V. Sridharan and D. R. Kaeli. Eliminating microarchitectural dependency from architectural vulnerability. In *Proceedings of the International Symposium on High-Performance Computer Architecture*, HPCA, 2009.

[235] V. Sridharan and D. R. Kaeli. Using hardware vulnerability factors to enhance AVF analysis. In *Proceedings of the IEEE international symposium on Computer architecture*, ISCA, pages 461–472, June 2010. ISBN 978-1-4503-0053-7.

[236] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi. Memory Errors in Modern Systems: The Good, The Bad, and The Ugly. *ACM SIGPLAN Notices*, 50:297–310, 2015. ISSN 0362-1340. doi:10.1145/2775054.2694348.

[237] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[238] J. E. Stine, I. D. Castellanos, M. H. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal. FreePDK: An open-source variation-aware design kit. In *IEEE International*

Conference on Microelectronic Systems Education (MSE), pages 173–174, 2007.

[239] D. Stow, Y. Xie, T. Siddiqua, and G. H. Loh. Cost-effective design of scalable high-performance systems using active and passive interposers. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017. doi:10.1109/ICCAD.2017.8203849.

[240] M. Sullivan, B. Zimmer, S. Hari, T. Tsai, and S. W. Keckler. An Analytical Model for Hardened Latch Selection and Exploration. SELSE, 2016.

[241] X. Sun, R. Nambiar, M. Melhorn, Y. Shoukry, and P. Nuzzo. DoS-Resilient Multi-Robot Temporal Logic Motion Planning. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 6051–6057, 2019. doi:10.1109/ICRA.2019.8794477.

[242] B. Sundaralingam, S. K. S. Hari, A. Fishman, C. Garrett, K. V. Wyk, V. Blukis, A. Millane, H. Oleynikova, A. Handa, F. Ramos, N. Ratliff, and D. Fox. curobo: Parallelized collision-free minimum-jerk robot motion generation, 2023.

[243] N. Sünderhauf, O. Brock, W. Scheirer, R. Hadsell, D. Fox, J. Leitner, B. Upcroft, P. Abbeel, W. Burgard, M. Milford, and P. Corke. The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5):405–420, 2018. doi:10.1177/0278364918770733.

[244] N. Tardella. Robots with high degrees of freedom face barriers to adoption. https://www.cobottrends.com/robots-with-high-degrees-of-freedom-face-barriers-to-adoption/, 2019.

[245] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2107–2115, 10 2017. doi:10.1109/ICCV.2017.230.

[246] Tesla. Future of driving. https://techxplore.com/news/2020-04-robots-heroes-war-coronavirus.html, 2020.

[247] J. Tilley. Automation, robotics, and the factory of the future. https://www.mckinsey.com/capabilities/operations/our-insights/automation-robotics-and-the-factory-of-the-future, 2017.

[248] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler. Nvbitfi: Dynamic fault injection for gpus. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 284–291, 2021. doi:10.1109/DSN48987.2021.00041.

[249] S. Tselonis and D. Gizopoulos. GUFI: A framework for GPUs reliability assessment. ISPASS, 2016. ISBN 9781509019526. doi:10.1109/ISPASS.2016.7482077.

[250] A. Tyagi, Y. Gan, S. Liu, B. Yu, P. Whatmough, and Y. Zhu. Thales: Formulating and estimating architectural vulnerability factors for dnn accelerators. *HPCA*, 2023. URL https://par.nsf.gov/biblio/10411383.

[251] Uber Advanced Technologies Group. We believe in the power of technology. https://www.uber.com/ca/en/atg/technology/, 2020.

[252] A. Vallero, S. Tselonis, D. Gizopoulos, and S. Di Carlo. Multi-faceted microarchitecture level reliability characterization for NVIDIA and AMD GPUs. VTS, pages 1–6. IEEE, 2018. ISBN 9781538637746. doi:10.1109/VTS.2018.8368665.

[253] J. van den Berg and M. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, 2004. doi:10.1109/ROBOT.2004.1307191.

[254] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, 2(4):1–13, jan 1998. ISSN 1086-7651. doi:10.1080/10867651.1997.10487480. URL https://doi.org/10.1080/10867651.1997.10487480.

[255] G. van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *Journal of Graphics Tools*, pages 1–13, 1998. ISSN 1086-7651.

[256] R. Venkatagiri, K. Swaminathan, C. Lin, L. Wang, A. Buyuktosunoglu, P. Bose, and S. Adve. Impact of software approximations on the resiliency of a video summarization system. In *Proceedings of the International Conference on Dependable Systems and Networks*, DSN, pages 598–609, 2018.

[257] R. Venkatagiri, K. Ahmed, A. Mahmoud, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve. gem5-approxilyzer: An open-source tool for

application-level soft error analysis. In *Proceedings of the International Conference on Dependable Systems and Networks*, DSN, pages 214–221, 2019. doi:10.1109/DSN.2019.00033.

[258] X. Vera, J. Abella, J.-A. Pineiro, A. Gonzalez, and R. Ronen. Selectively protecting a register file, 2008. Patent No. US20080155375A1, Filed Dec 20, 2006, Issued June 26, 2008.

[259] G. Verma and A. Swami. Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks. *Advances in Neural Information Processing Systems*, 32(NeurIPS), 2019.

[260] A. Veronesi, F. Dall'Occo, D. Bertozzi, M. Favalli, and M. Krstic. Exploring software models for the resilience analysis of deep learning accelerators: the nvdla case study. In *25th International Symposium on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, 2022.

[261] L. Wan, T. Alpcan, E. Viterbo, and M. Kuijper. Efficient error-correcting output codes for adversarial learning robustness. In *ICC 2022 - IEEE International Conference on Communications*, pages 2345–2350, 2022. doi:10.1109/ICC45855.2022.9839178.

[262] Z. Wan, K. Swaminathan, P.-Y. Chen, N. Chandramoorthy, and A. Raychowdhury. Analyzing and improving resilience and robustness of autonomous systems. In *ICCAD*, 2022. ISBN 9781450392174.

[263] J. Wang, T. Zhang, J. Song, N. Sebe, and H. Shen. A survey on learning to hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40 (04), 2018.

[264] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, PP, April 2020.

[265] J. Wang, T. Zhang, N. Ma, Z. Li, H. Ma, F. Meng, and M. Q.-H. Meng. A survey of learning-based robot motion planning. *IET Cyber-Systems and Robotics*, 2021. URL https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/csy2.12020.

[266] X. Wang, L. Bo, and F. Li. Adaptive wing loss for robust face alignment via heatmap regression. In *2019 IEEE International Conference on Computer Vision (ICCV)*, pages 6970–6980, 2019. doi:10.1109/ICCV.2019.00707.

[267] Washington State University. Smart home tests first elder care robot. https://www.sciencedaily.com/releases/2019/01/190114130913.htm, 2019.

[268] M. Wayland. Toyota's per-car profits lap detroit's big 3 automakers. https://www.detroitnews.com/story/business/autos/2015/02/22/toyota-per-car-profits-beat-ford-gm-chrysler/23852189/, 2015.

[269] Wikipedia. Programmable universal machine for assembly. https://en.wikipedia.org/wiki/Programmable_Universal_Machine_for_Assembly/, 2020.

[270] C. W. Wu. Designing communication systems via iterative improvement: error correction coding with bayes decoder and codebook optimized for source symbol error. *ArXiv:1805.07429*, 2018.

[271] W. Wu, C. Qian, S. Yang, Q. Wang, Y. Cai, and Q. Zhou. Look at boundary: A boundary-aware face alignment algorithm. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2129–2138, 2018.

[272] R. Xia, Y. Pan, H. Lai, C. Liu, and S. Yan. Supervised hashing for image retrieval via image representation learning. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, 2014.

[273] L. Xie, H. Chen, P. Qiu, and M. Zhang. The modified hamming bound for unequal error protection codes. In *IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions*, 2002. doi:10.1109/ICCCAS.2002.1180579.

[274] Z. Xu, B. Li, M. Geng, Y. Yuan, and G. Yu. Anchorface: An anchor-based facial landmark detector across large poses. *ArXiv:2007.03221*, 2020.

[275] T.-Y. Yang, Y.-H. Huang, Y.-Y. Lin, P.-C. Hsiu, and Y.-Y. Chuang. Ssr-net: A compact soft stagewise regression network for age estimation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, page 1078–1084. AAAI Press, 2018. ISBN 9780999241127.

[276] T.-Y. Yang, Y.-T. Chen, Y.-Y. Lin, and Y.-Y. Chuang. Fsa-net: Learning fine-grained structure aggregation for head pose estimation from a single image. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:1087–1096, 2019.

[277] X. Yang, L. Hou, Y. Zhou, W. Wang, and J. Yan. Dense label encoding for boundary discontinuity free rotation detection. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 15814–15824, 2021. doi:10.1109/CVPR46437.2021.01556.

[278] Y. Yang, X. Chen, and Y. Han. Dadu-cd: Fast and efficient processing-in-memory accelerator for collision detection. In *57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[279] A. Yazdanbakhsh, B. Akin, and K. K. Seshadri. An evaluation of edge tpu accelerators for convolutional neural networks. https://arxiv.org/abs/2102.10423, 2021.

[280] D. H. Yoon and M. Erez. Virtualized ecc: Flexible reliability in main memory. *IEEE Micro*, pages 11–19, 2011. doi:10.1109/MM.2010.103.

[281] E. Yoshida, K. Yokoi, and P. Gergondet. Online replanning for reactive robot motion: Practical aspects. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5927–5933, 2010. doi:10.1109/IROS.2010.5649645.

[282] C. Yu and S. Gao. Reducing collision checking for sampling-based motion planning using graph neural networks. In *Proceedings of the 35rd International Conference on Neural Information Processing Systems*, 2021.

[283] W. Zeng, W. Luo, S. Suo, A. Sadat, B. Yang, S. Casas, and R. Urtasun. End-to-end interpretable neural motion planner. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8652–8661, Los Alamitos, CA, USA, jun 2019. IEEE Computer Society. doi:10.1109/CVPR.2019.00886. URL https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00886.

[284] R. Zheng and M. C. Huang. Redundant memory array architecture for efficient selective protection. In *Proceedings of the International Symposium on Computer Architecture*, ISCA '17, 2017. ISBN 9781450348928. doi:https://doi.org/10.1145/3079856.3080213.

[285] X. Zhu, Z. Lei, X. Liu, H. Shi, and S. Li. Face alignment across large poses: A 3d solution. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 146–155, 06 2016.

Appendix A

Supplemental Material

A.1 Expected Error Derivation

This section explains the expected error equations used to compare BEL-U and BEL-J in Section 4.2. We first explain the encoding and decoding function used for BEL-U and derive the relation between the expected regression error and classification error for BEL-U. Then, we explain the encoding/decoding functions and expected error relation for BEL-J.

A.1.1 Preliminaries

Given a sample i drawn from a dataset with minimum label a and maximum label b, let $y_i \in [a,b]$ represent the target label for that sample. Assuming uniform quantization, the range of target labels can be quantized using $q : [a,b] \rightarrow \{1,2,...,N\}$ through Equation A.1.

$$q(y_i) = (y_i - a) * \frac{N-1}{b-a} + 1 \tag{A.1}$$

We define the encoding function $\mathscr{E} : \{1,2,...,N-1\} \rightarrow \{0,1\}^M$ to convert a target quantized level $Q_i \in \{1,2,...,N-1\}$ to a binary code $B_i \in \{0,1\}^M$. We further define the decoding function $\mathscr{D} : \{0,1\}^M \rightarrow [a,b]$ to convert the predicted binary code $\hat{B}_i$ to the predicted label $\hat{y}_i$.

Although the decoding functions used in this analysis predict the quantized

label and introduce quantization error, we do not include quantization error in the expected absolute error for our analysis as it is constant for both BEL-U and BEL-J. The expected value of absolute error between the target y and predicted labels $\hat{y}$ is used for the analysis as typically mean absolute error is used as the evaluation metric in regression problems.

Let us denote the error probability of a binary classifier C^k used to predict bit k in a binary code $B_i = \mathscr{E}(n)$ as $e_k(n)$, where n is the target quantized label Q_i. Then,

$$\begin{aligned} e_k(n) &= \mathbb{E}(|\hat{b}_i^k - b_i^k|) \\ &= Pr(\hat{b}_i^k = F) \text{ for target label } Q_i = n \text{ and target binary code } B_i = \mathscr{E}(n) \end{aligned} \tag{A.2}$$

where $\hat{b}_i^k = T$ indicates a correct binary classification by classifier C^k ($\hat{b}_i^k == b_i^k$) for sample i and $\hat{b}_i^k = F$ indicates an incorrect binary classification by classifier C^k ($\hat{b}_i^k \neq b_i^k$) for sample i.

A.1.2 Expected Error for BEL-U

Encoding and decoding functions: The encoding and decoding functions for BEL-U are defined as:

$$\mathscr{E}^{\text{BEL-U}}(Q_i) = b_i^1, b_i^2, .., b_i^{N-2}, \text{ where } b_i^k = \begin{cases} 1, & k < Q_i \\ 0, & \text{Otherwise} \end{cases} \tag{A.3}$$

$$\mathscr{D}^{\text{BEL-U}}(\hat{b}_i^1, \hat{b}_i^2, ..., \hat{b}_i^{N-2}) = \sum_{k=1}^{N-2} \hat{b}_i^k + 1 \tag{A.4}$$

Expected error: For target quantized label $Q_i = n$ ($n \in \{1, 2, ..., N-1\}$), ignoring the quantization error, the expected error between target y_i and predicted label $\hat{y}_i$

can be derived as:

$$\begin{aligned}
\mathbb{E}(|\hat{y}_i^{\text{BEL-U}} - y_i|) &= \mathbb{E}\left(\Big|\sum_{k=1}^{N-2}\left(\hat{b}_i^k + 1\right) - \sum_{k=1}^{N-2}\left(b_i^k + 1\right)\Big|\right) \\
&= \mathbb{E}\left(\Big|\sum_{k=1}^{N-2}(\hat{b}_i^k - b_i^k)\Big|\right) \\
&\leqslant \mathbb{E}\left(\sum_{k=1}^{N-2}|\hat{b}_i^k - b_i^k|\right) \\
&= \sum_{k=1}^{N-2}\mathbb{E}\,|\hat{b}_i^k - b_i^k| \\
&= \sum_{k=1}^{N-2} e_k(n) \text{ (using Equation A.2)}
\end{aligned} \tag{A.5}$$

For a uniform distribution of target labels in the range $[1, N-1]$, the expected error can be derived as:

$$\mathbb{E}(|\hat{y}^{\text{BEL-U}} - y|) \leqslant \frac{1}{N-1}\sum_{n=1}^{N-1}\sum_{k=1}^{N-2} e_k(n) \tag{A.6}$$

A.1.3 Expected Error for BEL-J

Encoding and decoding functions: For target quantized label $Q_i \in \{1, 2, ..., N-1\}$, BEL-J encoding requires $\frac{N}{2}$ bits/binary classifiers. The encoding for BEL-J can be defined as:

$$\mathscr{E}^{\text{BEL-J}}(Q_i) = b_i^1, b_i^2, .., b_i^{\frac{N}{2}}, \text{where } b_i^k = \begin{cases} 1, & \frac{N}{2} - Q_i < k \leqslant N - Q_i \\ 0, & \text{Otherwise} \end{cases} \tag{A.7}$$

Similarly, the decoding functions for BEL-J can be defined as:

$$\begin{aligned} \mathscr{D}^{\text{BEL-J}}(\hat{B}_i) &= Tl(\hat{B}_i) + Tf(\hat{B}_i) + Tc \\ \text{where, } Tl(\hat{B}_i) &= -\max_{k\in\{1\ldots\frac{N}{2}\}} k\hat{b}_i^k \\ Tf(\hat{B}_i) &= \max_{k\in\{1\ldots\frac{N}{2}\}} \left(\frac{N}{2}-k+1\right)\hat{b}_i^k, Tc = \frac{N}{2} \end{aligned} \tag{A.8}$$

In Equation A.8, $Tl()$ finds the location of the last occurrence of "1" in the predicted binary code $\hat{B}_i$. Similarly, $Tf()$ finds the location of the first occurrence of "1" in the binary code $\hat{B}_i$. Figure A.1 gives examples of binary codes for label $Q_i \in \{1,2,...,7\}$ and the corresponding values of the different terms in Equation A.8. For example, for label $Q_i = 3$, the binary code is "0111". Here, the last occurrence of "1" is at position 4, and $Tl = -4$. Similarly, the first occurrence of "1" is at position 2, and $Tf = (4+1)-2 = 3$.

Expected error: For BEL-J code, binary classifiers $(C^1, C^2, ..., C^{\frac{N}{2}})$ are used. For a given input sample i, an error in any of the binary classifiers' outputs $(\hat{b}_i^1, \hat{b}_i^2, ..., \hat{b}^{\frac{N}{2}})$ will result in an error between $Tf(\hat{B}_i)/Tl(\hat{B}_i)$ and $Tf(B_i)/Tl(B_i)$ in Equation A.8.

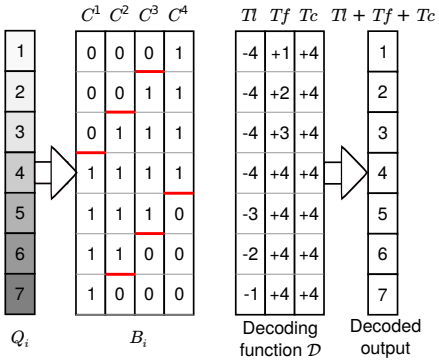


Figure A.1: Encoding and Decoding functions' output for BEL-J approach and label $y \in [1, N-1]$, where $N = 8$. Decoding function's output is calculated using $y' = Tl + Tf + Tc$, where $Tl = -\max_{k\in\{1\ldots\frac{N}{2}\}} k\hat{b}_i^k$, $Tf = \max_{k\in\{1\ldots\frac{N}{2}\}} \left(\frac{N}{2}-k+1\right)\hat{b}_i^k$, and $Tc = \frac{N}{2}$
.

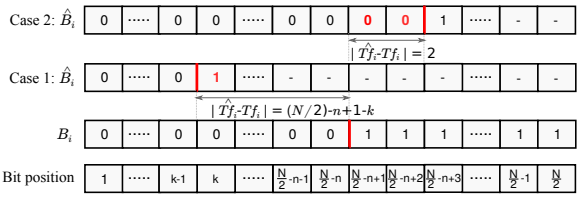


Figure A.2: Effect of classifier error on $\hat{Tf}_i - Tf_i$ for label $Q_i = n$. Case 1 and case 2 represent erroneous outputs. $0/1$ highlighted in red color represents an error in the classifier's output. "-" represents error/no error in both cases.

We refer to $Tf(\hat{B}_i)$ and $Tl(\hat{B}_i)$ as $\hat{Tf}_i$ and $\hat{Tl}_i$ (predicted binary code), and $Tf(B_i)$ and $Tl(B_i)$ as Tf_i and Tl_i (target binary code) for brevity. Expected value of the absolute error can be further expanded as:

$$\begin{aligned}
\mathbb{E}(|\hat{y}_i^{\text{BEL-J}} - y_i|) &= \mathbb{E}(|\hat{Tf}_i + \hat{Tl}_i + Tc - (Tf_i + Tl_i + Tc)|) \\
&= \mathbb{E}(|(\hat{Tf}_i - Tf_i) + (\hat{Tl}_i - Tl_i)|) \\
&\leqslant \mathbb{E}(|\hat{Tf}_i - Tf_i| + |\hat{Tl}_i - Tl_i|) \\
&= \mathbb{E}(|\hat{Tf}_i - Tf_i|) + \mathbb{E}(|\hat{Tl}_i - Tl_i|)
\end{aligned} \tag{A.9}$$

Thus, the sum of the expected error of $Tf()$ and $Tl()$ is the upper bound of the label's expected error. Further, we derive the relation between binary classifiers' error probabilities and $\mathbb{E}(|\hat{Tf}_i - Tf_i|)$ and $\mathbb{E}(|\hat{Tl}_i - Tl_i|)$.

We consider $Q_i = n$, where $1 \leqslant n \leqslant \frac{N}{2}$ for our derivation. In such a case, $Tf_i = n$ and $Tl_i = -\frac{N}{2}$. However, as the code is symmetric around $Q_i = \frac{N}{2}$, it can be shown that the derived equation for $\mathbb{E}\,|\hat{y}_i - y_i|$ can be used for $1 \leqslant Q_i \leqslant N-1$.

1. Derivation of $\mathbb{E}\,|\hat{Tf}_i - Tf_i|$: As shown in Equation A.8, $Tf()$ finds the location k of the first occurrence of "1" in the binary sequence. In the case of an erroneous binary sequence, the position of the first occurrence of "1" might shift, which results in an error between $\hat{Tf}_i$ and Tf_i. Figure A.2 shows examples of the correct and erroneous outputs of classifiers for label $Q_i = n$. For label $Q_i = n$, $b_i^k = 0$ for $k \in \{1, 2, ..., \frac{N}{2} - n\}$ and $b_i^k = 1$ for $k \in \{\frac{N}{2} - n + 1, \frac{N}{2} - n + 2, ..., \frac{N}{2}\}$.

For case 1, error in a classifier $C^k, k \in \{1, 2, ..., \frac{N}{2} - n\}$ is considered, where $b_i^k = 0$ and $\hat{b}_i^k = 1$. For $k \in \{1, 2, ..., \frac{N}{2} - n\}$, an error at classifier C^k will result

in erroneous $\hat{Tf}_i$ only if all proceeding classifiers are correct, since if any of the proceeding classifier z is incorrect, i.e. $\hat{b}_i^z = 1$, then the location of the first occurrence of "1" will be shifted to z, and any error in the following classifiers will not affect the value of $\hat{Tf}$. Such a case ($\hat{b}_i^1 = T, \hat{b}_i^2 = T, ..., \hat{b}_i^{k-1} = T, \hat{b}_i^k = F, \hat{b}_i^{k+1} = T/F, ..., \hat{b}_i^{\frac{N}{2}} = T/F$) considers a total of $2^{\frac{N}{2}-k}$ combinations out of $2^{\frac{N}{2}}$ for $k \in \{1, 2, ..., \frac{N}{2} - n\}$. Assuming that the binary classifiers are mutually independent, the error value and the probability of this combination can be shown to be:

$$|\hat{Tf}_i - Tf_i| = \left(\frac{N}{2} - n + 1 - k\right) \tag{A.10}$$

$$\begin{aligned} Pr(\hat{b}_i^1 = T, \hat{b}_i^2 = T, ..., \hat{b}_i^{k-1} = T, \hat{b}_i^k = F) &= Pr(\hat{b}^1 = T) Pr(\hat{b}_i^2 = T) ... Pr(\hat{b}_i^{k-1} = T) Pr(\hat{b}^k = F) \\ &= \left(\prod_{j=1}^{k-1} (1 - e_j(n))\right) \cdot e_k(n) \end{aligned} \tag{A.11}$$

The above term considers combinations ($b'^1 = T, b'^2 = T, ..., b'^{k-1} = T, b'^k = F, b'^{k+1} = T/F, ..., b'^{\frac{N}{2}=T/F}$) for $k \in \{1, 2, ..., \frac{N}{2} - n\}$, which constitutes to a total of $\sum_{k=1}^{\frac{N}{2}-n} 2^{\frac{N}{2}-k}$ combinations out of $2^{\frac{N}{2}}$.

For case 2, error in a classifier $C^k, k \in \{\frac{N}{2} - n + 1, \frac{N}{2} - n + 2, ..., \frac{N}{2}\}$ is considered, where $b_i^k = 1$ and $\hat{b}_i^k = 0$. We consider a combination ($\hat{b}_i^1 = T, \hat{b}_i^2 = T, ..., \hat{b}_i^{\frac{N}{2}-n} = T, \hat{b}_i^{\frac{N}{2}-n+1} = F, ..., \hat{b}_i^{k-1} = F, \hat{b}_i^k = T, \hat{b}_i^{k+1} = T/F, ..., \hat{b}_i^{\frac{N}{2}=T/F}$). For this case, the position of the first occurrence of "1" will be moved to k, which will result in erroneous $\hat{Tf}_i$. Such a case would cover $2^{\frac{N}{2}-k}$ combinations out of $2^{\frac{N}{2}}$ for $k \in \{\frac{N}{2} - n + 1, \frac{N}{2} - n + 2, ..., \frac{N}{2}\}$. The error value and the probability of this combination can be shown to be:

$$|\hat{Tf}_i - Tf_i| = \left(k - (\frac{N}{2} - n + 1)\right) \tag{A.12}$$

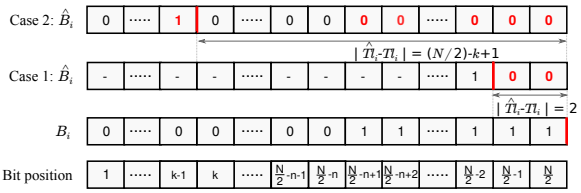


Figure A.3: Effect of classifier error on $\hat{Tl}_i - Tl_i$ for label $Q_i = n$. Case 1 and case 2 represent erroneous outputs. $0/1$ highlighted in red color represents an error in the classifier's output. "-" represents error/no error in both cases.

$$Pr(\hat{b}_i^1 = T, \hat{b}_i^2 = T, ..., \hat{b}_i^{\frac{N}{2}-n} = T, \hat{b}_i^{\frac{N}{2}-n+1} = F, ..., \hat{b}_i^{k-1} = F, \hat{b}_i^k = T) = \Big(\prod_{j=1}^{\frac{N}{2}-n}(1-e_j(n))\Big) \cdot \Big(\prod_{j=\frac{N}{2}-n+1}^{k-1} e_j(n)\Big) \cdot \Big(1-e_k(n)\Big) \tag{A.13}$$

The above term considers combinations $(\hat{b}_i^1 = T, \hat{b}_i^2 = T, ..., \hat{b}_i^{\frac{N}{2}-n} = T, \hat{b}_i^{\frac{N}{2}-n+1} = F, ..., \hat{b}_i^{k-1} = F, \hat{b}_i^k = T, \hat{b}_i^{k+1} = T/F, ..., \hat{b}_i^{\frac{N}{2}=T/F})$, which constitutes to a total of $\sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} 2^{\frac{N}{2}-k}$ combinations out of $2^{\frac{N}{2}}$ for $k \in \{\frac{N}{2}-n+1, \frac{N}{2}-n+2, ..., \frac{N}{2}\}$.

Combining Equation A.10 to Equation A.13, the expected value of $|\hat{Tf}_i - Tf_i|$ can be derived as:

$$\begin{aligned}\mathbb{E}(|\hat{Tf}_i - Tf_i|) &= \sum_{k=1}^{\frac{N}{2}-n}\Big(\frac{N}{2}-n+1-k\Big)\cdot\Big(\prod_{j=1}^{k-1}(1-e_j(n))\Big)\cdot e_k(n) \\ &\quad + \sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}}\Big(k-(\frac{N}{2}-n+1)\Big)\cdot\Big(\prod_{j=1}^{\frac{N}{2}-n}(1-e_j(n))\Big)\cdot\Big(\prod_{j=\frac{N}{2}-n+1}^{k-1}(e_j(n))\Big)\cdot\Big(1-e_k(n)\Big) \\ &= \sum_{k=1}^{\frac{N}{2}-n}\Big(\frac{N}{2}-n+1-k\Big)\cdot e_k(n)\cdot\Big(\prod_{j=1}^{k-1}(1-e_j(n))\Big) + \sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}}\Big(\prod_{j=\frac{N}{2}-n+1}^{k} e_j(n)\Big)\end{aligned} \tag{A.14}$$

The first term in Equation A.14 covers $\sum_{k=1}^{\frac{N}{2}-n} 2^{\frac{N}{2}-k}$ combinations and the second term considers $\sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} 2^{\frac{N}{2}-k}$ combinations. Adding one combination where all the classifiers are correct, Equation A.14 considers all of the possible combinations $2^{\frac{N}{2}}$ to find expected value of $|\hat{Tf}_i - Tf_i|$.

2. Derivation of $\mathbb{E}\,|\hat{Tl}_i - Tl_i|$: As shown in Equation A.8, $Tl()$ finds the location

k of the last occurrence of "1" in the binary sequence. In the case of an erroneous binary sequence, the position of the last occurrence of "1" might shift, which results in an erroneous value of $\hat{T}l_i$. Figure A.3 shows examples of correct and erroneous outputs of classifiers for label $Q_i = n$.

For case 1, an error in a classifier $C^k, k \in \{\frac{N}{2}-n+1, \frac{N}{2}-n+2, ..., \frac{N}{2}\}$ is considered, where $b_i^k = 1$ and $\hat{b}_i^k = 0$. We consider a combination $(\hat{b}_i^{\frac{N}{2}} = F, \hat{b}_i^{\frac{N}{2}-1} = F, ..., \hat{b}_i^{k+1} = F, \hat{b}_i^k = T, \hat{b}_i^{k-1} = T/F, ..., \hat{b}_i^1 = T/F)$. For this case, position of the last occurrence of "1" will be moved to k, which will result in erroneous $\hat{T}l_i$. Such a case would cover 2^{k-1} combinations out of $2^{\frac{N}{2}}$. The error value and the probability of this combination can be shown to be:

$$|\hat{T}l_i - Tl_i| = \left(\frac{N}{2} - k\right) \tag{A.15}$$

$$Pr(\hat{b}_i^{\frac{N}{2}} = F, \hat{b}_i^{\frac{N}{2}-1} = F, ..., \hat{b}_i^{k+1} = F, \hat{b}_i^k = T) = \Big(\prod_{j=k+1}^{\frac{N}{2}} e_j(n)\Big) \cdot (1 - e_k(n)) \tag{A.16}$$

The above term considers combinations $(\hat{b}_i^{\frac{N}{2}} = F, \hat{b}_i^{\frac{N}{2}-1} = F, ..., \hat{b}_i^{k+1} = F, \hat{b}_i^k = T, \hat{b}_i^{k-1} = T/F, ..., \hat{b}_i^1 = T/F)$ for $k \in \{\frac{N}{2}-n+1, \frac{N}{2}-n+2, ..., \frac{N}{2}\}$, which constitutes to a total of $\sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} 2^{k-1}$ combinations out of $2^{\frac{N}{2}}$.

For case 2, an error in a classifier $C^k, k \in \{1, 2, ..., \frac{N}{2}-n\}$ is considered, where $b_i^k = 0$ and $\hat{b}_i^k = 1$. We consider a combination $(\hat{b}_i^{\frac{N}{2}} = F, ..., \hat{b}_i^{\frac{N}{2}-n+1} = F, \hat{b}_i^{\frac{N}{2}-n} = T, ..., \hat{b}_i^{k+1} = T, \hat{b}_i^k = F, \hat{b}_i^{k-1} = T/F, ..., \hat{b}_i^1 = T/F)$. For this case, position of the last occurrence of "1" will be moved to k, which will result in erroneous $\hat{T}l_i$. Such a case would cover 2^{k-1} combinations out of $2^{\frac{N}{2}}$. The error value and the probability of this combination can be shown to be:

$$|\hat{T}l_i - Tl_i| = \left(\frac{N}{2} - k\right) \tag{A.17}$$

$$\begin{aligned} Pr(\hat{b}_i^{\frac{N}{2}} = F, ..., \hat{b}_i^{\frac{N}{2}-n+1} = F, \hat{b}_i^{\frac{N}{2}-n} = T, ..., \hat{b}_i^{k+1} = T, \hat{b}_i^k = F) = \\ \Big(\prod_{j=\frac{N}{2}-n+1}^{\frac{N}{2}} e_j(n)\Big) \cdot \Big(\prod_{j=k+1}^{\frac{N}{2}-n} (1 - e_j(n))\Big) \cdot (e_k(n)) \end{aligned} \tag{A.18}$$

The above term considers combinations ($\hat{b}_i^{\frac{N}{2}} = F, ..., \hat{b}_i^{\frac{N}{2}-n+1} = F, \hat{b}_i^{\frac{N}{2}-n} = T, ..., \hat{b}_i^{k+1} = T, \hat{b}_i^{k} = F, \hat{b}_i^{k-1} = T/F, ..., \hat{b}_i^{1} = T/F$) for $k \in \{1, 2, ..., \frac{N}{2} - n\}$, which constitutes to a total of $\sum_{k=1}^{\frac{N}{2}-n} 2^{k-1}$ combinations out of $2^{\frac{N}{2}}$.

Combining Equation A.15 to Equation A.18, the expected value of $|\hat{Tl}_i - Tl_i|$ can be derived as:

$$\begin{aligned}
\mathbb{E}(|\hat{Tl}_i - Tl_i|) = & \sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} \Big(\frac{N}{2} - k\Big) \cdot \Big(\prod_{j=k+1}^{\frac{N}{2}} e_j(n)\Big) \cdot (1 - e_k(n)) \\
& + \sum_{k=1}^{\frac{N}{2}-n} \Big(\frac{N}{2} - k\Big) \cdot \Big(\prod_{j=\frac{N}{2}-n+1}^{\frac{N}{2}} e_j(n)\Big) \cdot \Big(\prod_{j=k+1}^{\frac{N}{2}-n} (1 - e_j(n))\Big) \cdot (e_k(n)) \\
& = \sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} \Big(\prod_{j=k}^{\frac{N}{2}} e_j(n)\Big) + \Big(\prod_{j=\frac{N}{2}-n+1}^{\frac{N}{2}} e_j(n)\Big) \cdot \sum_{k=1}^{\frac{N}{2}-n} \Big(\prod_{j=k}^{\frac{N}{2}-n} (1 - e_j(n))\Big)
\end{aligned} \tag{A.19}$$

The first term in Equation A.19 covers $\sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} 2^{k-1}$ combinations and the second term considers $\sum_{k=1}^{\frac{N}{2}-n} 2^{k-1}$ combinations. Adding one combination where all the classifiers are correct, Equation A.19 considers all of the possible combinations $2^{\frac{N}{2}}$ to find expected value of $|\hat{Tl}_i - Tl_i|$.

Combining Equation A.9, Equation A.14, and Equation A.19, the expected value of error for $Q_i = n$ in terms of classifiers' error probabilities can be derived as:

$$\begin{aligned}
\mathbb{E}(\hat{y}_i^{\text{BEL-J}} - y_i) \leqslant & \sum_{k=1}^{\frac{N}{2}-n} \Big(\frac{N}{2} - n + 1 - k\Big) \cdot e_k(n) \cdot \Big(\prod_{j=1}^{k-1} (1 - e_j(n))\Big) + \sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} \Big(\prod_{j=\frac{N}{2}-n+1}^{k} e_j(n)\Big) \\
& + \sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} \Big(\prod_{j=k}^{\frac{N}{2}} e_j(n)\Big) + \Big(\prod_{j=\frac{N}{2}-n+1}^{\frac{N}{2}} e_j(n)\Big) \cdot \sum_{k=1}^{\frac{N}{2}-n} \Big(\prod_{j=k}^{\frac{N}{2}-n} (1 - e_j(n))\Big)
\end{aligned} \tag{A.20}$$

As the binary code is symmetric around $\frac{N}{2}$ as shown in Figure A.1, the expected errors for label $y_i \in [1, \frac{N}{2}]$ can be mirrored to find expected errors for label $y_i \in [\frac{N}{2}, N-1]$. For a uniform distribution of target labels in the range $[1, N-1]$, the

expected error can be derived as:

$$\mathbb{E}(\hat{y}^{\text{BEL-J}} - y) \leqslant \frac{1}{N-1} \sum_{n=1}^{N-1} \Bigg[\sum_{k=1}^{\frac{N}{2}-n} \left(\frac{N}{2} - n + 1 - k \right) \cdot e_k(n) \cdot \left(\prod_{j=1}^{k-1} (1 - e_j(n)) \right) + \sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} \left(\prod_{j=\frac{N}{2}-n+1}^{k} e_j(n) \right)$$
$$+ \sum_{k=\frac{N}{2}-n+1}^{\frac{N}{2}} \left(\prod_{j=k}^{\frac{N}{2}} e_j(n) \right) + \left(\prod_{j=\frac{N}{2}-n+1}^{\frac{N}{2}} e_j(n) \right) \cdot \sum_{k=1}^{\frac{N}{2}-n} \left(\prod_{j=k}^{\frac{N}{2}-n} (1 - e_j(n)) \right) \Bigg] \quad \text{(A.21)}$$

We also verify the equation by comparing the expected value of error based on Equation A.20 for $Q_i \in \{1, 2, ..., N-1\}$ with the expected error calculated by $100,000$ random samples of binary sequences for the same error probabilities $e_k(n)$. Figure A.4 compares the expected error from Equation A.20 and measured from statistical samples, and validates error upper bounds calculated using Equation A.20 and Equation A.21.

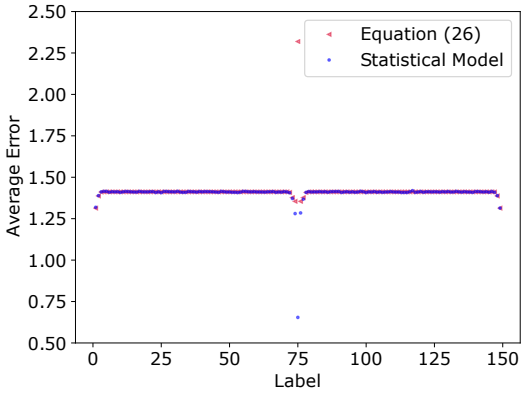


Figure A.4: Comparison of expected value of error from Equation A.20 and random samples for given error probabilities of the classifiers.

A.2 Evaluation of BEL Design Functions

We propose multiple combinations of encoding, decoding, and loss functions that can be used with BEL. In Tables A.1- A.11, we show the effect of each combination of encoding, decoding, and loss function on the error of the model. Although general

trends exist and some combinations perform consistently well across datasets, the optimal combination varies based on the dataset.

Table A.1: Comparison of BEL design parameters on MAE for head pose estimation with BIWI dataset and ResNet50 feature extractor (LFH1).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	3.38	3.65	-	-	-	-
GEN-EX	BCE	**3.37**	3.64	5.11	8.02	4.76	7.53
GEN	BCE	3.38	3.65	5.16	8.16	4.99	7.73
GEN-EX	CE	4.22	3.55	3.88	4.08	4.09	5.50
GEN	CE	4.25	3.62	3.93	4.06	4.39	5.48
GEN-EX	L2	3.56	3.93	3.66	3.59	5.99	4.21

Table A.2: Comparison of BEL design parameters on MAE for head pose estimation with 300LP/AFLW2000 datasets and ResNet50 feature extractor (LFH2).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	4.78	4.84	-	-	-	-
GEN-EX	BCE	**4.77**	4.84	5.43	5.09	4.94	7.84
GEN	BCE	4.78	4.87	5.11	5.05	5.15	8.54
GEN-EX	CE	4.93	5.04	5.04	4.97	4.79	5.64
GEN	CE	5.07	5.17	5.13	5.10	4.99	5.62
GEN-EX	L2	5.05	5.18	5.19	5.09	5.17	5.07

Table A.3: Comparison of BEL design parameters on MAE for head pose estimation with BIWI dataset and RAFA-Net feature extractor (LFH3).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	3.47	3.16	-	-	-	-
GEN-EX	BCE	3.46	**3.12**	3.30	3.35	3.80	5.75
GEN	BCE	3.49	3.14	3.62	3.78	4.44	5.83
GEN-EX	CE	3.82	3.91	3.52	3.49	3.98	3.98
GEN	CE	3.92	4.09	3.62	3.65	4.35	4.28
GEN-EX	L2	3.72	3.60	4.31	4.29	6.61	18.69

Table A.4: Comparison of BEL design parameters on MAE for head pose estimation with 300LP/AFLW2000 datasets and RAFA-Net feature extractor (LFH4).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	3.94	4.00	-	-	-	-
GEN-EX	BCE	**3.90**	3.93	4.19	4.12	4.39	9.17
GEN	BCE	3.93	3.94	4.21	4.25	4.53	9.21
GEN-EX	CE	4.55	4.62	4.34	4.53	4.45	5.12
GEN	CE	4.68	4.75	4.46	4.61	4.63	5.29
GEN-EX	L2	4.45	5.87	5.11	9.34	10.43	17.89

Table A.5: Comparison of BEL design parameters on NME for facial landmark detection with COFW dataset and HRNetV2-W18 feature extractor (FLD1).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	3.47	3.45	-	-	-	-
GEN-EX	BCE	3.45	3.43	3.42	3.41	3.47	4.28
GEN	BCE	3.46	3.45	3.43	3.47	3.66	4.43
GEN-EX	CE	3.37	3.37	3.38	3.41	**3.34**	3.69
GEN	CE	3.44	3.44	3.44	3.49	3.57	3.69
GEN-EX	L1	3.44	3.41	3.45	3.47	3.41	4.52

Table A.6: Comparison of BEL design parameters on NME for facial landmark detection with 300W dataset and HRNetV2-W18 feature extractor (FLD2).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	3.5	3.49	-	-	-	-
GEN-EX	BCE	3.48	3.46	3.43	3.42	3.38	4.71
GEN	BCE	3.50	3.49	3.45	3.45	3.55	4.78
GEN-EX	CE	3.40	**3.36**	3.37	3.41	3.37	3.62
GEN	CE	3.50	3.45	3.45	3.51	3.59	3.65
GEN-EX	L1	3.41	3.39	3.49	3.67	3.43	4.04

Table A.7: Comparison of BEL design parameters on NME for facial landmark detection with WFLW dataset and HRNetV2-W18 feature extractor (FLD3).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	4.62	4.54	-	-	-	-
GEN-EX	BCE	4.6	4.51	4.43	4.38	4.37	7.18
GEN	BCE	4.62	4.53	4.44	4.42	4.55	7.14
GEN-EX	CE	4.36	4.34	4.36	**4.33**	4.34	5.15
GEN	CE	4.46	4.44	4.47	4.47	4.56	4.83
GEN-EX	L1	4.39	4.42	4.47	4.47	4.45	4.74

Table A.8: Comparison of BEL design parameters on NME for facial landmark detection with AFLW dataset and HRNetV2-W18 feature extractor (FLD4).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	1.51	1.52	-	-	-	-
GEN-EX	BCE	1.50	1.50	**1.47**	**1.47**	1.49	1.52
GEN	BCE	1.51	1.52	1.50	1.49	1.54	1.55
GEN-EX	CE	1.48	**1.47**	**1.47**	**1.47**	**1.47**	**1.47**
GEN	CE	1.52	1.51	1.51	1.51	1.52	1.52
GEN-EX	L1	**1.47**	**1.47**	1.48	1.48	1.48	1.59

Table A.9: Comparison of BEL design parameters on MAE for age estimation with MORPH-II dataset and ResNet50 feature extractor (AE1).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	2.32	**2.27**	-	-	-	-
GEN-EX	BCE	2.30	2.29	2.35	2.49	2.45	2.99
GEN	BCE	2.28	2.28	2.34	2.51	2.54	3.07
GEN-EX	CE	2.55	2.54	2.75	2.65	2.63	12.33
GEN	CE	2.60	2.58	2.61	2.66	2.61	3.10
GEN-EX	L1	2.30	2.30	2.32	2.30	2.32	2.29

Table A.10: Comparison of BEL design parameters on MAE for age estimation with AFAD dataset and ResNet50 feature extractor (AE2).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	3.13	3.15	-	-	-	-
GEN-EX	BCE	3.14	3.16	3.32	3.35	3.28	3.34
GEN	BCE	3.13	3.19	3.41	3.44	3.41	3.52
GEN-EX	CE	3.26	3.29	3.38	3.44	3.40	3.30
GEN	CE	3.36	3.34	3.42	3.47	3.40	3.45
GEN-EX	L1	3.13	3.12	**3.11**	3.12	3.13	3.13

Table A.11: Comparison of BEL design parameters on MAE for end-to-end learning of self-driving car steering with PilotNet dataset and feature extractor (PN).

Decoding function	Loss function	Encoding function					
		U	J	B1JDJ	B2JDJ	HEXJ	HAD
BEL-J/BEL-U	BCE	4.34	3.91	-	-	-	-
GEN-EX	BCE	4.57	4.20	4.83	4.96	5.29	10.12
GEN	BCE	4.37	3.95	3.51	3.61	4.01	10.00
GEN-EX	CE	4.30	4.16	4.99	5.87	5.39	87.17
GEN	CE	3.15	**3.11**	3.14	3.21	3.64	6.20
GEN-EX	L1	4.10	4.11	4.34	4.34	4.11	5.09