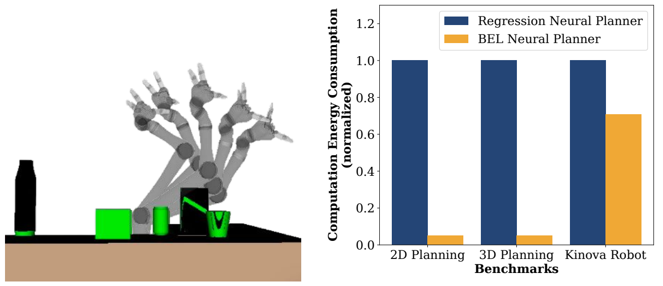
Neural Encodings for Energy-Efficient Motion Planning

Jocelyn Zhao[1], Deval Shah[1,2], Tor M. Aamodt
Department of Electrical and Computer Engineering
University of British Columbia, Vancouver, BC, Canada
{jzhao42,devalshah,aamodt}@ece.ubc.ca

***Abstract*— Neural motion planners can increase motion planning quality and, by reducing collision detection computations, improve runtime. However, when profiled on an accelerator-rich hardware system, neural planning contributes to more than 50% of the runtime, and 33% of the computation energy consumption, motivating the design of compute- and energy-efficient neural planners. In this work, we propose a neural planner using Binary Encoded Labels (BEL), where a set of binary classifiers are used instead of a typical regression network. Compared to conventional regression-based neural planners, the proposed BEL neural planner reduces neural planning (inference) computation and collision detection checks while maintaining equal or higher motion planning success rate across various motion planning benchmarks. This computation reduction can improve the computation energy efficiency of neural planning by $1.4\times - 21.4\times$. Finally, we demonstrate the trade-offs between collision detection and neural planning computation to maximize energy efficiency for different hardware configurations.**



(a) Example of a 6DOF path (b) Runtime comparison

Fig. 1: (a) shows an example of a Kinova 6DOF motion planning path, and (b) compares the average computation requirement of neural planning for baseline regression-based neural planning and the proposed BEL neural planning for 2D, 3D and 6DOF motion planning problems.

I. Introduction

Robotic motion planning is used to find a collision-free and safe path for a robot to reach its goal position. Motion planning is used for various applications, from navigation in self-driving cars to full-body movement planning for humanoid robots. Motion planning is a computationally intensive task with strict realtime latency constraints. Several works have focused on algorithm [17], [5], [54], [42], [16] and/or hardware [53], [32], [36], [49], [2], [45], [27] optimizations for realtime and energy-efficient motion planning.

More recently, several works have explored neural planners for motion planning to improve the motion planning success rate achieved in fixed runtime budget [54], [42], [17], [11]. These approaches often combine deep learning with traditional motion planning algorithms. However, these approaches primarily focus on improving the motion planning quality and not on improving the end-to-end computational efficiency of the motion planning. Further, these works do not consider the relationship between a neural planner and the computing system, which can significantly affect the total motion planning runtime and computational power consumption.

We first profile motion planning using MPNet [42], a neural motion planning approach, on a CPU-GPU system, where collision detection takes $23\times$ more time than neural planning. We then analyze MPNet running on an accelerator-rich system using microarchitectural simulation to study the efficiency of neural planning executed on an EdgeTPU [10] and collision detection accelerated by specialized collision detection unit (CDU) hardware [45]. Even on such a system neural planning still consumes $\sim 50\%$ of the runtime and $\sim 33\%$ of total power consumption. Thus, **designing energy-efficient neural planners can lead to significant reductions in runtime and power consumption.**

This paper introduces a Binary-Encoded Label (BEL)-based neural planner as an alternative to regression-based neural planners. Prior works have explored regression by binary classification using binary-encoded labels [30], [38], [44] for deep regression networks and have demonstrated improved accuracy due to ensemble diversity [47] and added redundancy in label representation [12]. However, the application of regression by binary classification to robotic motion planning and its impact on key performance metrics remain unexplored. In this work, we adapt BEL for neural motion planning to enhance overall motion planning quality and computation efficiency. We show that the proposed BEL neural planner improves accuracy, increases motion planning success rate, and reduces computation requirements. Additionally, we analyze its effectiveness across different motion planning phases (Section III) and demonstrate that the BEL

[1]Contributed equally.
[2]Work performed while affiliated with UBC. Preliminary results on BEL neural planner were published in the author's doctoral dissertation [43].
*This paper was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

neural planner achieves a higher success rate during the initial planning phase compared to direct regression.

The BEL neural planner reduces computation by enabling the use of a smaller and sparser neural network while maintaining the motion planning success rate and quality. Fig. 1 compares the neural planning inference computation of a regression-based neural planner and the BEL neural planner across different motion planning problems. The BEL neural planner reduces neural planning computation by $1.3\times - 90\times$ ($10.5\times$ geometric mean) while achieving equivalent motion planning success rates (not shown). Finally, we show that various configurations of the BEL neural planner provide different trade-offs between the computation requirements of neural planning inference and collision detection. We evaluate different hardware accelerator configurations for neural planning and collision detection and show that the energy efficiency of these underlying hardware accelerators affects the choice of neural planning model.

II. Background and Related Work

In this section, we briefly summarize existing works on learning-based motion planning approaches and provide relevant background on binary encoded labels for regression neural network. Further, we summarize existing works on hardware acceleration of motion planning, collision detection, and neural network inference.

A. Learning-based Motion Planning

Learning-based motion planning is an active area of research in robotics. Different types of learning-based motion planning approaches include end-to-end neural planners [21], [42], [15], [29], [55], combinations of neural network and conventional motion planners [34], [9], [41], [26], [42], [54], [17], and reinforcement learning-based motion planners [3], [25], [14]. These approaches have shown significant improvement in the motion planning capability and runtime for various motion planning problems.

Motion Planning Network (MPNet): Qureshi et al. [42] proposed a motion planning network to replace sampling in motion planning. Fig. 2 illustrates their approach. Here, $p_i \in \mathbb{R}^d$ represents a sampled pose in the configuration space of a robot with d degrees of freedom. The current step $\hat{p}_t$, goal position p_{goal}, and encoded environment occupancy information ENet(E_{obs}) are provided to the neural planner, which predicts the next step $\hat{p}_{t+1}$ in the trajectory. The neural planner is iteratively executed until a collision-free path between the start and goal is found. A trajectory between the start and goal positions of the robot consists of M milestones $\{p_{start}, \hat{p}_2, ..., \hat{p}_t, \hat{p}_{t+1}, ..., \hat{p}_{M-1}, p_{goal}\}$. Collision detection is then used to find the feasibility of the trajectory provided by the neural planner (not shown). Prior works used a regression network-based neural planners to predict real-values ($\hat{p}_{t+1}$). These networks are trained by minimizing the mean squared or absolute error between the target p_{t+1} and predicted values $\hat{p}_{t+1}$, where the target values can be generated using an oracle planner. Increasing the accuracy of the neural planning phase helps find a path that is more likely to be collision-free, reducing computation costs.

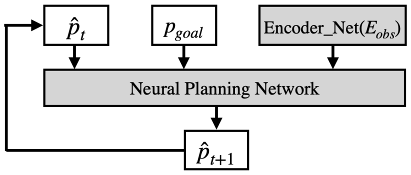
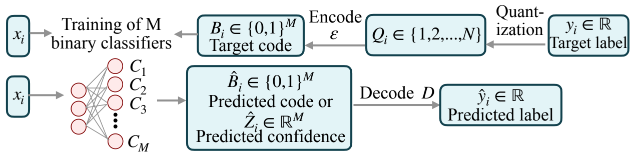


Fig. 2: Neural Planning Network for motion planning [42].



x_i
Training of M binary classifiers
$B_i \in \{0,1\}^M$ Target code
Encode ε
$Q_i \in \{1,2,...,N\}$
Quant-ization
$y_i \in \mathbb{R}$ Target label
x_i
C_1
C_2
C_3
C_M
$\hat{B}_i \in \{0,1\}^M$ Predicted code or $\hat{Z}_i \in \mathbb{R}^M$ Predicted confidence
Decode D
$\hat{y}_i \in \mathbb{R}$ Predicted label

Fig. 3: Training and inference workflow for a BEL-based regression network

B. Binary Encoded Labels

Shah et al. [44] proposed using Binary Encoded Labels (BEL) to improve the regression error for a given task. The architecture for BEL-based networks differs from a typical regression network in the choice of regressor: An N-bit binary code is predicted instead of a scalar, real-valued output logit. Training and inference of a BEL-based network is summarized in Fig. 3. During training (top), a real-valued target y_i is converted to an M-bit binary code B using an encoding function. This code is used as a target to train M binary classifiers fed by output x_i of a backbone network. During inference (bottom), the BEL-based neural network predicts a code, which is then converted to a real-valued prediction using a decoding function. Shah et al. [44], proposed several suitable encoding and decoding functions for regression. Prior works [30], [38], [44] have shown that the use of multiple binary classifiers improves regression accuracy due to added redundancy and error correction capability. These works, however, have focused on improving the regression Fzz for a range of regression tasks, while the BEL network's suitability for neural planning and its impact on overall success rate computation has remained unexplored.

C. Motion Planning Acceleration

Motion planning is a computationally intensive problem, that must be executed in real-time to ensure the safety of the robot and its surroundings. Further, the energy efficiency of motion planning is important due to hardware constraints (e.g., mobile robots). Several acceleration approaches have been explored for motion planning, including on GPUs [4], [19], Specialized Accelerators [35], [32], [53], [2], [27], [45], and FPGAs [1], [46]. Collision detection is the most time and energy-consuming part of sampling-based motion planning algorithms [4], [39]. Hence, several specialized accelerators have focused on the acceleration of collision detection [35], [49], [2], [45], [53]. These specialized low-power collision detection accelerators reduce the collision detection runtime

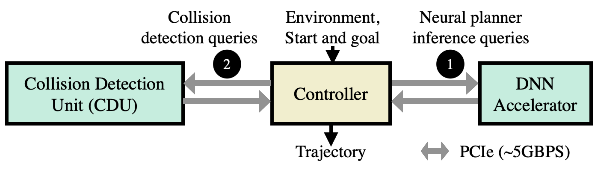

Collision detection queries
Environment, Start and goal
Neural planner inference queries
Collision Detection Unit (CDU)
Controller
DNN Accelerator
2
1
Trajectory
PCIe (~5GBPS)

Fig. 4: Block diagram of an accelerator-rich learning-based motion planning system

by orders of magnitude compared to CPUs. In this work, we focus on learning-based motion planning approaches that consist of neural sampling and collision detection. Fig. 4 represents the block diagram of an accelerator-rich system for learning-based motion planning. Here, a Deep Neural Network Accelerator (DNNAccel) is used for neural planning, and Collision Detection Units (CDUs) are used to execute collision detection for the robot. A controller (e.g., simple Arm Cortex core) executes the motion planning algorithm, and offloads neural planning ❶ and collision detection ❷ to the accelerators. These accelerators are connected through high bandwidth communication links (e.g. PCIe [37]).

D. Neural Network Acceleration

Hardware acceleration of deep neural network inference has been studied extensively. Several works focus on designing dataflow and memory systems around an array of multiply-accumulate units [13], [10], [40], [33], [8], [7]. Another approach for acceleration is to reduce the number of parameters and computation of a neural network, which includes quantization of parameters, sparsity, and designing smaller networks [22], [20], [24]. Different types of sparse neural network hardware accelerators are proposed by prior works [23], [52], [51], [18], [31], [6], [48] to take advantage of static sparsity introduced by pruning of the network during training and/or dynamic sparsity introduced by dropout and activation layers (e.g., ReLU).

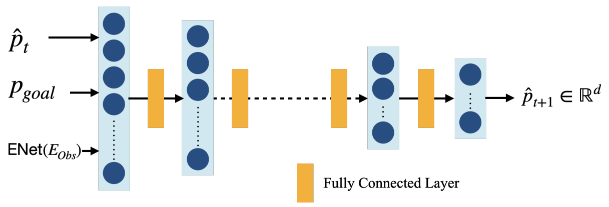
Srivastava et al. [50] first proposed dropout as a regularization technique in neural networks used during training to prevent overfitting. Each node in the hidden layer of the neural network is dropped with probability dp, determined by the dropout rate. Recent works have leveraged dropout during inference to introduce stochasticity to the network's output. Dropout-induced dynamic sparsity can be leveraged by specialized hardware accelerators for reducing computation and power consumption [48], [31].

III. BEL Neural Planner

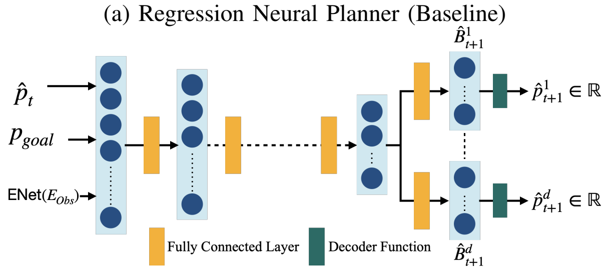
In this section, we provide details of the proposed Binary-Encoded Label-based neural planner. We modify a learning-based motion planning framework MPNet [42] in this work (explained in Section II-A), which uses a traditional regression network to predict real-valued motion planning milestones. In this work, we aim to explore the use of Binary Encoded Labels (explained in Section II-B) for neural planning and understand its impact on motion planning success rate, trajectory quality, and computation requirements.

TABLE I: Example of Unary code used for Binary-Encoded Labels for four quantization levels. p^i_{t+1} represents the real-valued target label, and B represents the target binary code.

p^i_{t+1}	B^i_{t+1}
$p^i_{t+1} \in [0,1)$	000
$p^i_{t+1} \in [1,2)$	001
$p^i_{t+1} \in [2,3)$	011
$p^i_{t+1} \in [3,4)$	111



(a) Regression Neural Planner (Baseline)



(b) BEL Neural Planner

Fig. 5: (a) and (b) represent the architecture of the baseline regression-based and proposed BEL neural planners.

Fig. 5a represents baseline regression neural planner. Given the current position, goal, and environment, the neural planner predicts a d-dimensional next step $\hat{p}_{t+1} \in \mathbb{R}^d$ for motion planning. Here d represents the number of DOFs of the robot. This network is trained by minimizing $||\hat{p}_{t+1} - p_{t+1}||_2$, where p_{t+1} is the target next step provided by an oracle planner. Fig. 5b represents proposed BEL neural planner. The final layer of the baseline neural planner is modified to predict $d \times N$ (N dimensional code for each DOF) values. Note that since the size (i.e., number of parameters) of the last layer is a small fraction of total size of the network, the increase in the output size results in less than 1% increase in the neural planner's size.

Algorithm 1 is used for training the BEL neural planner. The target label p^i_{t+1} for i^{th} DOF is converted to a binary code B^i_{t+1} (Line 2) using the Encoding function based on Table I. The network predicts d codes $\hat{B}^i_{t+1}$, where each code is of size N (Line 3). The BEL neural planner is trained by minimizing the binary classification loss between $\hat{B}^i_{t+1}$ and target binary code B^i_{t+1} (Line 4). Algorithm 2 is used for inference/planning using BEL neural planner. BEL neural planner predicts d codes, where each code $\hat{B}^i_{t+1}$ is an N-dimensional predicted code (Line 1). $\hat{B}^i_{t+1}$ is passed through a decoding function to find real-valued $\hat{p}^i_{t+1}$ (Line 2).

Decoding Function: We use an expected correlation-based decoding function proposed by [44]. In this method, a code matrix C of size $(N+1) \times N$, where N is the number of bits/classifiers used for the binary-encoded label,

Algorithm 1 Training BEL Neural Planner (one epoch)

```
Input: Untrained Neural_Planner, Trained ENet, Training_Set, Encoding Function
Output: Trained Neural_Planner;
1: for $(p_t, p_{goal}, \text{ENet}(E_{obs}), p_{t+1} \in$ Training_set do
2:     $B_{t+1}$ = [Encoding($p^1_{t+1}$), Encoding($p^2_{t+1}$),..., Encoding($p^d_{t+1}$)]
3:     $\hat{B}_{t+1}$ = [$\hat{B}^1_{t+1}, \hat{B}^2_{t+1}, ..., \hat{B}^d_{t+1}$] = Neural_Planner(($p_t, p_{goal}$, ENet($E_{obs}$))
4:     Loss = Binary classification ($B_{t+1}$,$\hat{B}_{t+1}$)
5:     Neural_Planner.optimize(Loss)
6: end for
```

Algorithm 2 Inference with BEL Neural Planner

```
Input: Trained Neural_Planner, Decoding Function, ($\hat{p}_t, p_{goal}$, ENet($E_{obs}$)
Output: $\hat{p}_{t+1}$;
1: $\hat{B}_{t+1}$ = Neural_Planner(($\hat{p}_t, p_{goal}$, ENet($E_{obs}$))
2: for $i \in 1, 2, ..., d$ do $\hat{p}^i_{t+1}$ = [Decoding($\hat{B}^i_{t+1}$) for $i$in[1, ..., $d$] ]
3: end for
```

and the target label is in the range $[0, N+1)$. Each row $C_{k,:}$ in this matrix represents the binary code for label $x_{t+1} \in [k, k+1)$. For example, Table I represents a code matrix for unary encoding function. The real values of the output of the neural planner $\hat{B}_{t+1}$ represents the confidence of binary classifiers. We can reduce the quantization error by using the output confidence of binary classifiers and expected correlation for decoding, as shown by prior works [44]. The decoding function is defined as:

$$\hat{p}^i_{t+1} = \sum_{k=1}^{N+1} k \times \frac{e^{\hat{B}^i_{t+1} \cdot C_{k,:}}}{\sum_{j=1}^{N+1} e^{\hat{B}^i_{t+1} \cdot C_{j,:}}} \quad (1)$$

BEL Neural Planning The motion planning algorithm used in this work, MPNet, consists of two phases: global planning and iterative local planning. During phase-1 (global planning phase), the neural planner finds a full trajectory between the start and goal positions, which is checked for collision. If one or more segments in this path are found to be colliding, phase-2 neural planning (local replanning) is done for each segment by using the start and end positions of this segment. MPNet proposed to use dropout [50] for stochastic inference to explore a different path during phase-2. Use of dropout enables sampling next position from a distribution instead of deterministic value for a given input.

We observe that BEL neural planner has significantly higher success rate compared to regression neural planner during the phase-1 (Section IV). The phase-2 of the MPNet algorithm, however, relies on dropout for exploring different paths between two positions. We found that BEL neural planner does not explore many different paths during phase-2 as it is more robust to sparsity introduced by dropout, even at higher dropout rates (see Fig. 7). This is the most evident when motion planning for a 6 DOF robotic arm, thus for that benchmark only, we propose the use of a smaller regression neural planner (< 1M parameters) to generate a more diverse set of milestones during phase-2.

IV. Evaluation

This section describes our experimental methodology and a detailed evaluation of BEL neural planner in terms of its impact on motion planning success rate and computation.

TABLE II: Neural planner architectures evaluated in this work. Param is the number of weight parameters, and compute is the number of FLOPs per inference.

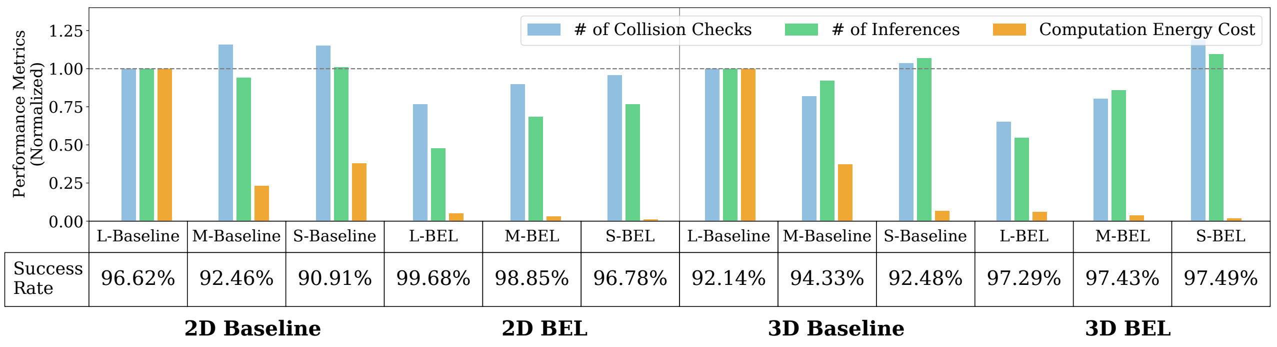
	Name	Num. of layers	Dropout	Param ($\times 10^3$)	Compute ($\times 10^3$FLOPs)
2D and 3D point navigation	L-Baseline	12	50%	3754	1878
	M-Baseline	9	50%	1550	776
	S-Baseline	6	50%	264	133
	L-BEL	12	90%	3755	379
	M-BEL	9	90%	1552	158
	S-BEL	6	80%	266	56
Kinova robot arm motion planning	Baseline_dp0p5	9	50%	2036	1018
	BEL_dp0p5	9	50%	2044	1022
	BEL_dp0p9	9	90%	2044	204
	Replanner	5	50%	887	443

A. Methodology

Motion Planning Benchmarks: We evaluate three planning problems: 2D and 3D point navigation, and motion planning for a 6 DOF Kinova Gen2 robot arm following the same experimental methodology as MPNet [42]. For the point path planning problems, the neural planner is trained on 4000 paths generated by RRT* [28] across 100 workspaces and evaluated on 10 workspaces not seen during training, with 2000 random pairs of start and goal positions in each workspace. The 6DOF Kinova robot neural planner is trained on 1000 paths generated by RRT* [28] in 10 simulated environments and evaluated on 100 randomly generated start and goal positions per environment. We also validate our neural planner on a real Kinova robot using point clouds collected by an Intel Realsense Depth Camera D435.

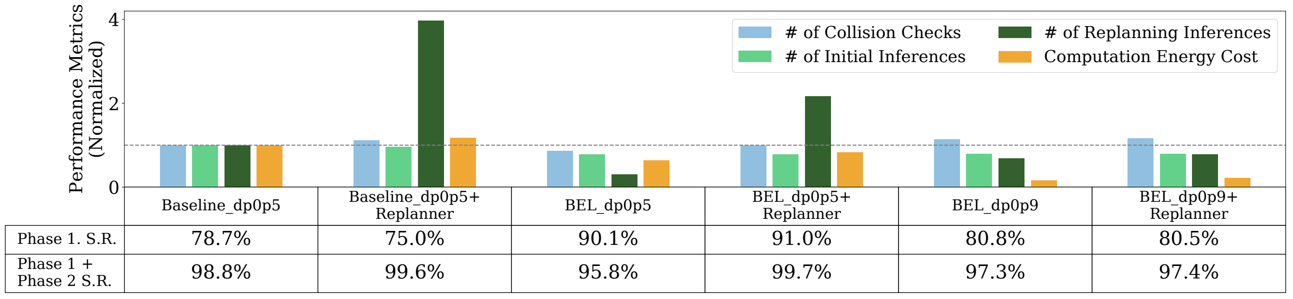
Neural Planner Model Architecture: Obstacle information is collected as a point cloud and passed into an encoder network, ENet, with three layers. For 2D and 3D navigation, we use a pre-trained encoder network provided by MPNet [42], and neural planners are trained for 400 epochs. For the neural planner for the 6DOF Kinova robot, the encoder (ENet) and neural planner are trained end-to-end for 200 epochs. All planners apply Dropout [50] during inference in all layers except the last and genereate d-dimensional trajectory milestones. The BEL neural planner uses a unary code of size $N = 40$ (i.e., length of the binary code). Table II summarizes model size, dropout rate, and FLOPs per inference for each of the three different neural planner architectures explored in this work. Here, L/M/S prefix represents the size of the neural planner. The dropout rate is selected based on empirical evaluation.

Hardware Accelerators: For the baseline configuration of the collision detection unit and neural planning hardware, we consider a collision detection unit (CDU) proposed by [45] and neural network accelerator EdgeTPU [10]. We use the microarchitectural CDU simulator provided by [45] to estimate the number of cycles needed for a single collision detection test between an oriented bounding box or robotic arm and the environment and determine the approximate energy per collision detection from the reported power consumption results. For neural planning, we use



	L-Baseline	M-Baseline	S-Baseline	L-BEL	M-BEL	S-BEL	L-Baseline	M-Baseline	S-Baseline	L-BEL	M-BEL	S-BEL
Success Rate	96.62%	92.46%	90.91%	99.68%	98.85%	96.78%	92.14%	94.33%	92.48%	97.29%	97.43%	97.49%

(a) Path planning benchmarks for 2D and 3D environments, with computation metrics (collision checks, inferences, and energy) normalized to the L-Baseline.



	Baseline_dp0p5	Baseline_dp0p5+ Replanner	BEL_dp0p5	BEL_dp0p5+ Replanner	BEL_dp0p9	BEL_dp0p9+ Replanner
Phase 1. S.R.	78.7%	75.0%	90.1%	91.0%	80.8%	80.5%
Phase 1 + Phase 2 S.R.	98.8%	99.6%	95.8%	99.7%	97.3%	97.4%

(b) Path planning for a 6DOF Kinova robot arm (normalized to the Baseline metrics). S.R. indicates the success rate. Phase 1 is the global planning stage and Phase 2 is the replanning phase, as described in Section III.

Fig. 6: Comparison of motion planning quality and computation for the baseline and BEL neural planners across different experimental setups.

inference FLOPs (including dynamic sparsity due to dropout) to estimate power consumption. We assume a hardware DNN accelerator capable of leveraging dynamic sparsity introduced by dropout, with a performance of 1 TOPS/W. Note that such hardware support may not yet be available on any commercial platform. However, there is growing research on enabling hardware support for dynamic sparsity [23], [52], [51], [18], [31], [6], [48].

B. Experimental Results

This section provides a detailed evaluation of BEL neural planner. We further evaluate the impact of dropout rate on BEL neural planner for stochastic inference. Finally, we discuss impact of hardware acceleration for neural planning and collision detection on the choice of neural planner.

*1) **BEL Neural Planning**:* We evaluate the Binary Encoded Labels neural planner described in Section III. Our findings indicate that BEL-based networks perform fewer collision detection checks and neural planning inferences compared to the baseline regression-based neural planners (MPNet [42]) with equivalent or higher motion planning success rates.

Fig. 6a compares the proposed BEL neural planner's motion planning success rate and computation requirements with the baseline for 2D and 3D path planning. In the 2D environment, the best-performing BEL neural planner has a $20\times$ reduction in computation energy consumption while having a 3.06% higher success rate than the best-performing baseline. Similarly, the BEL neural planner has a $21.4\times$ reduction in energy consumption and 3.16% higher success rate. Our findings show that a smaller BEL neural planner can maintain competitive performance compared against its larger counterpart: S-Baseline has a 5.71% lower success rate, while S-BEL only degrades 2.9%. In the 2D and 3D environments, BEL neural planners demonstrate robustness to increased sparsity, achieving higher success rates and lower computational costs even at a dropout rate of `dp=0.9`.

Fig. 6b evaluates the motion planning success rate and computation requirements for the baseline and BEL neural planners for a 6DOF Kinova Gen2 robotic arm across different neural planner configurations. Phase 1 refers to the planning success rate (S.R.) and Phase 2 refers to the replanning success rate (S.R.), as defined in Section III. Neural planners with higher success rates in Phase 1 require fewer collision checks and inferences. BEL_dp0p5 has the lowest number of inferences and collision checks, however, it obtains a success rate of 95.8%. We attribute the reduced computation energy consumption to the higher Phase 1 success rate and reduced replanning inferences. BEL_dp0p9 has a success rate of 97.3 but incurs higher collision checks and inferences. We observe reduced energy consumption due to the increased sparsity in the BEL_dp0p9 network. The baseline network at `dp=0.9` is unable to produce a trajectory. As noted in Section III, BEL is less effective in Phase 2 due to dropout-induced sparsity in stochastic inference (see Section IV-B.2), leading to lower success rates for a pure BEL-based planner. To leverage the computational benefits of the BEL neural planner, it is desirable to have a high initial success rate and only introduce the required

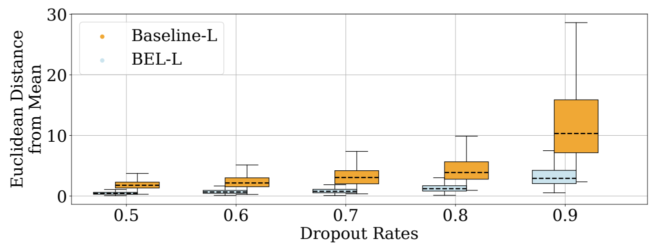


Fig. 7: Variation in the euclidean distance between the generated milestones and the mean for a given start and goal position at varying dropout rates for the baseline regression network and the bel network.

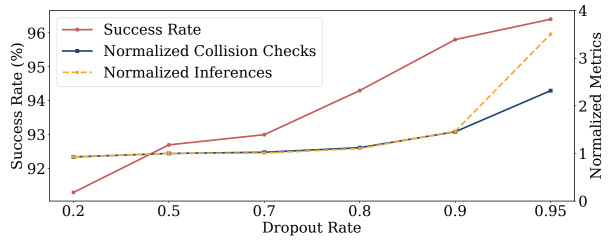


Fig. 8: Ablation study on the effect of dropout rate on the motion planning success rate and computational requirements. Collision checks and inferences are normalized to the results for `dp=0.5`.

stochasticity during Phase 2. Based on this insight, we propose a hybrid BEL neural planner, using BEL in Phase 1 and small regression network (Replanner from Table II) during Phase 2. Implementing the Replanner with the BEL networks during the replanning phase improves the success rate to 99.7% while reducing overall computation energy consumption compared to the baseline. We attribute this to the Replanner's ability to generate more diverse milestones. Compared to Baseline + Replanner, the implementation of BEL_dp0p5+Replanner and BEL_dp0p9 results in $1.4\times$ and $5.3\times$ reduction in energy consumption respectively.

*2) **Stochasticity with Dropout in BEL Neural Planner**:* Prior works on learning-based motion planning [42] proposed to use Dropout [50] to introduce stochasticity in the sampled path. Thus, the neural planner can explore multiple paths between two points until a feasible solution is found. For a 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 real-value, which makes it more robust to sparsity introduced by dropout. As illustrated in Fig. 8, the BEL neural planner requires a higher dropout rate to attain the same or higher success rate compared to baseline neural planner in the point planning benchmarks. A high dropout rate reduces the computation required for inference, which can be leveraged by hardware platforms supporting dynamic sparsity.

Fig. 7 shows the variation in generated milestones at different dropout rates for the baseline and BEL neural planners. The BEL neural planner consistently produces more similar milestones than the baseline regression network and remains robust at high dropout rates. Thus BEL requires a higher dropout rate for stochastic inference. While dropout can introduce variety, an overly aggressive dropout rate negatively affects the ability of the neural planner to represent the embedded information. A dropout level exceeding 0.9 leads to a significant increase in number of inferences and collision checks while yielding only a marginal improvement in success rate (Fig. 8).

*3) **Hardware-aware Neural Planner Selection** :* We observe that different BEL neural planners provides motion planning success rate in $\pm1\%$ range (Fig. 6). Intuitively, the smallest neural planner is the best choice. However, there exists a trade-off between the computation costs for collision detection and neural network inferences. A smaller neural planner saves the neural planning computation at the expense of increased exploration, requiring higher costs for collision detection. In contrast, a larger neural planner decreases the number of collision detection runs needed for motion planning, though it incurs higher neural planning costs. Neural planner selection depends on the underlying hardware accelerators and their energy consumption.

Different factors impact the energy consumption of a robot-environment collision detection check, such as the environment, robot's geometric representation and granularity of collision detection for a path. The energy consumption for neural planning inference depends upon several factors, such as numeric representation and support for sparsity. Different hardware accelerators provide varying performance in terms of TOPS/W. For the hardware accelerator configuration used in this work (explained in Section IV-A), M-BEL results in the lowest computation power consumption for 2D path planning. In contrast, if the power consumption of neural network accelerator increases by $4\times$, S-BEL results in the lowest total power consumption. Similarly, if the power consumption of collision detection accelerator increases by $4\times$, L-BEL results in the lowest total power consumption.

V. Conclusion

This work focuses on designing accurate and compute-efficient neural planners for various motion planning problems. We propose to use binary classification-based neural planners using Binary-Encoded Labels (BEL) instead of traditional regression-based neural planners. The proposed BEL neural planner improves the motion planning success rate and results in a $1.4\times - 21.4\times$ reduction in energy consumption due to increased accuracy and higher introduced sparsity. BEL neural planners are more robust and thus have less dependence on replanning, suggesting that they may improve the success rates of dynamic policy networks while retaining its computational benefits. Furthermore, we examine how the underlying hardware and its energy consumption for collision detection and neural planning influence the selection of the neural planner.

REFERENCES

[1] N. Atay and B. Bayazit, “A motion planning processor on reconfigurable hardware,” in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2006, pp. 125–132.

[2] 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.

[3] 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)*, 2018, pp. 3175–3182.

[4] J. Bialkowski, S. Karaman, and E. Frazzoli, “Massively parallelizing the RRT and the RRT,” in *International Conference on Intelligent Robots and Systems*, 2011, pp. 3513–3518.

[5] R. Bohlin and L. Kavraki, “Path planning using lazy prm,” in *IEEE International Conference on Robotics and Automation*, 2000, pp. 521–528.

[6] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, “Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[7] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, “Dadiannao: A machine-learning supercomputer,” in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-47. USA: IEEE Computer Society, 2014, p. 609622. [Online]. Available: https://doi.org/10.1109/MICRO.2014.58

[8] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, “Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks,” in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, pp. 262–263.

[9] 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)*, 2020, pp. 7485–7492.

[10] Coral AI, “AI google coral tpu overview and products,” 2020. [Online]. Available: {https://dls.ieiworld.com/IEIWEB/MARKETING_MATERIAL/2021_catalog/0-10_AI_Google_TPU_overview_%26_products.pdf}

[11] M. Dalal, J. Yang, R. Mendonca, Y. Khaky, R. Salakhutdinov, and D. Pathak, “Neural mp: A generalist neural motion planner,” 2024. [Online]. Available: https://arxiv.org/abs/2409.05864

[12] T. G. Dietterich and G. Bakiri, “Solving Multiclass Learning Problems via Error-Correcting Output Codes,” *Journal of Artificial Intelligence Research*, vol. 2, pp. 263–286, 1995.

[13] J. et al., “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2017, pp. 1–12.

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

[15] A. Fishman, A. Murali, C. Eppner, B. N. Peele, B. Boots, and D. Fox, “Motion policy networks,” in *Conference on Robot Learning*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253098016

[16] 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*, vol. 39, no. 5, pp. 543–567, 2020.

[17] 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*, 2014, pp. 2997–3004.

[18] Y. Gao, B. Zhang, X. Qi, and H. K.-H. So, “Dpacs: Hardware accelerated dynamic neural network pruning through algorithm-architecture co-design,” in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 237251. [Online]. Available: https://doi.org/10.1145/3575693.3575728

[19] R. Gayle, P. Segars, M. Lin, and D. Manocha, “Path planning for deformable robots in complex environments,” in *Robotics: Science and Systems*, 2005, pp. 225–232.

[20] 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*, vol. abs/2103.13630, 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[21] 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)*, pp. 1–7, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:4048147

[22] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1510.00149

[23] X. He, S. Pal, A. Amarnath, S. Feng, D.-H. Park, A. Rovinski, H. Ye, Y. Chen, R. Dreslinski, and T. Mudge, “Sparse-tpu: Adapting systolic arrays for sparse matrices,” in *Proceedings of the 34th ACM International Conference on Supercomputing*, ser. ICS ’20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3392717.3392751

[24] 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*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[25] J. Huh and D. D. Lee, “Efficient sampling with q-learning to guide rapidly exploring random trees,” *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3868–3875, 2018.

[26] 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)*, 2020, pp. 9535–9541.

[27] 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*, 2022, p. 909914.

[28] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *International Journal of Robotic Research - IJRR*, vol. 30, pp. 846–894, 06 2011.

[29] 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*, ser. NIPS’18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 87478758.

[30] L. Li and H.-T. Lin, “Ordinal regression by extended binary classification,” in *Proceedings of the 19th International Conference on Neural Information Processing Systems*, 2006, pp. 865–872.

[31] Z. Li, J. Li, T. Chen, D. Niu, H. Zheng, Y. Xie, and M. Gao, “Spada: Accelerating sparse matrix multiplication with adaptive dataflow,” in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 747761. [Online]. Available: https://doi.org/10.1145/3575693.3575706

[32] 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*, ser. DAC. Association for Computing Machinery, 2018.

[33] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, “Cambricon: An instruction set architecture for neural networks,” in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA ’16. IEEE Press, 2016, p. 393405. [Online]. Available: https://doi.org/10.1109/ISCA.2016.42

[34] 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)*, 2020, pp. 10 605–10 611.

[35] 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*, ser. ASAP. IEEE, 2019, pp. 185–188.

[36] 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.

[37] 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*, ser. SIGCOMM ’18. Association for Computing Machinery, 2018, p. 327341.

[38] 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.

[39] J. Pan and D. Manocha, “Gpu-based parallel collision detection for fast motion planning,” *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 187–200, 2012.

[40] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, “Scnn: An accelerator for compressed-sparse convolutional neural networks,” vol. 45, no. 2, p. 2740, jun 2017.

[41] N. Prez-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)*, 2018, pp. 5897–5902.

[42] 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*, vol. PP, pp. 1–19, 08 2020.

[43] D. Shah, “Energy-efficient acceleration for autonomous robotics,” Ph.D. dissertation, The University of British Columbia, 2024.

[44] D. Shah, Z. Y. Xue, and T. M. Aamodt, “Label encoding for regression networks,” in *International Conference on Learning Representations*, April 2022. [Online]. Available: https://openreview.net/pdf?id=8WawVDdKqlL

[45] D. Shah, N. Yang, and T. M. Aamodt, “Energy-efficient realtime motion planning,” 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

[46] X. Shi, L. Cao, D. Wang, L. Liu, G. You, S. Liu, and C. Wang, “HERO: Accelerating Autonomous Robotic Tasks with FPGA,” ser. IROS, 2018, pp. 7766–7772.

[47] Y. Song, Q. Kang, and W. P. Tay, “Error-Correcting Output Codes with Ensemble Diversity for Robust Learning in Neural Networks,” *AAAI*, 2021.

[48] 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)*, 2019, pp. 108–113.

[49] 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.

[50] 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*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[51] F. Tu, Y. Wang, L. Liang, Y. Ding, L. Liu, S. Wei, S. Yin, and Y. Xie, “Sdp: Co-designing algorithm, dataflow, and architecture for in-sram sparse nn acceleration,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 109–121, 2023.

[52] G. Xiao, C. Yin, T. Zhou, X. Li, Y. Chen, and K. Li, “A survey of accelerating parallel sparse linear algebra,” *ACM Comput. Surv.*, vol. 56, no. 1, aug 2023. [Online]. Available: https://doi.org/10.1145/3604606

[53] Y. Yang, X. Chen, and Y. Han, “Dadu-cd: Fast and efficient processing-in-memory accelerator for collision detection,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[54] 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.

[55] 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)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2019, pp. 8652–8661. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00886