ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

UDC 004.94

doi: 10.32620/reks.2025.4.08

Andrii MEDVID, Vitaliy YAKOVYNA

Lviv Polytechnic National University, Lviv, Ukraine

PER-LINK COLLISION DEPTH PREDICTION FOR REDUNDANT MANIPULATORS IN OPERATIONAL ENVIRONMENTS

*The **subject matter** of this study is a collision checking for redundant robotic manipulators operating in variable environments, which remains a significant computational bottleneck in motion planning. The **goal** of this study is to improve computational efficiency of collision checking for multi-joint robotic manipulators in sampling-based motion planning, while preserving functional safety. This is achieved by developing and evaluating a learning-based method that predicts per-link penetration depth and serves as a statistical pre-filter rather than a replacement for exact collision checking. The **tasks** are as follows: 1) to propose a novel input representation that fuses the manipulator's kinematic state with localized geometric context extracted from the environment via voxel grids; 2) to design and implement a hybrid neural network architecture combining a fully-connected projection layer with a Kolmogorov-Arnold Network (KAN); 3) to train the network on a large, procedurally generated dataset of diverse collision scenarios; and 4) to evaluate the model's regression accuracy, classification performance, and computational speedup over a direct physics-based checker. The following **results** were obtained: the trained model achieves high regression accuracy with a low Mean Squared Error of 0.000148 on the test set; the model achieves promising classification results with a per-link recall of 93.01%, which is an important indicator for its use as a pre-filter capable of screening out the majority of hazardous states; computational speedup - performance benchmarks for a batch of 8192 states show that the proposed approach, including data preparation and inference, is approximately 3.7 times faster than a direct physics-based checker. **Conclusions.** The scientific novelty of results obtained is as follows: 1) a neural network architecture combining fully-connected and Kolmogorov–Arnold Network layers is proposed for predicting per-link collision depth of a redundant manipulator; 2) integration of kinematic and voxel-based geometric features into a unified input representation for accurate collision estimation. The proposed method effectively serves as a pre-filter for sampling-based planners, reducing the number of expensive collisions checks and accelerating the overall motion planning process.*

Keywords: *Robotic manipulator; collision check; voxel grids; Kolmogorov–Arnold Networks.*

1. Introduction

1.1. Motivation

Collision checking remains a dominant computational bottleneck in motion planning for redundant robotic manipulators operating in varying operational environments. In sampling-based pipelines (e.g., RRT-family methods [1]) and trajectory planners alike, tens of thousands of candidate arm states or dense waypoint sequences must be evaluated for collisions. Physics-based collision queries (e.g., using the Bullet library [2]) provide reliable answers but are computationally expensive. A fast, learning-based estimator that screens large batches of states – with conservative use as a pre-filter – can substantially reduce the number of expensive exact checks, thereby shortening planning time.

Earlier work by the authors [3] addressed a simpler setting: self-collision detection for a robot with a 7-DoF arm without an environment map. Joint angles were encoded as sine/cosine pairs and fed to a KAN (Kolmogorov–Arnold Network) [4] classifier, achieving about 98.5% binary accuracy on self-collision labels. The present study extends that line of research to the realistic case where the arm operates in a workspace with obstacles: the environment is explicitly represented and the prediction target changes from a binary self-collision indicator to per-link penetration depth. To supply the model with localized geometric context at inference time, each link (and the wrist-mounted tool) is paired with a small axis-aligned local voxel grid of collision depths with the map and the base, and with the link pose expressed as a relative position and a unit quaternion with a canonical sign. This design permits batched processing on modern accelerators and amortizes repeated voxel queries through caching voxel collision depths.

This research focuses on accelerating the collision checking process for a robotic arm in varied environments by using neural networks to evaluate batches of potential states.

The scientific novelty consists in: (i) proposing a per-link input construction that fuses local, axis-aligned voxel grids of collision depth with relative link poses; (ii) formulating a general, batched per-link depth estimation

approach that targets the screening stage of motion planning. The specific combination of localized voxel evidence, pose features, and a compact FC (fully connected) + KAN architecture for per-link depth regression to the best of our knowledge has not been previously reported for redundant manipulators in mapped workspaces; finally, (iii) the work introduces a scalable per-link collision screening paradigm that explicitly targets the pre-filtering stage of sampling-based motion planners rather than replacing exact collision checking.

In this context, the proposed method focuses on fixed-base redundant manipulators with a wrist-mounted tool; environment states may vary, but are captured at data-generation time through local voxel depth patches. The model predicts per-link collision depths that can be aggregated into state- or trajectory-level risk scores, supporting planner heuristics. Importantly, the method explicitly designed as a pre-filter, not a replacement for exact collision checking.

For clarity, in this work the term "redundant manipulator" refers to kinematic redundancy, i.e., the presence of more degrees of freedom than strictly required to achieve a given end-effector pose. The proposed method does not exploit redundancy for fault tolerance, joint failure recovery, or null-space optimization. Instead, redundancy is relevant because it increases the dimensionality of the configuration space, which amplifies the computational burden of collision checking. The proposed approach remains applicable to non-redundant manipulators, but is particularly motivated by high-DoF systems.

Figure 1 illustrates the redundant seven-joint xArm7 manipulator by UFactory, along with the rotational joint limits. The robot model with the manipulator mounted on a robotic platform was used both to generate the training data and to experimentally evaluate the proposed approach.

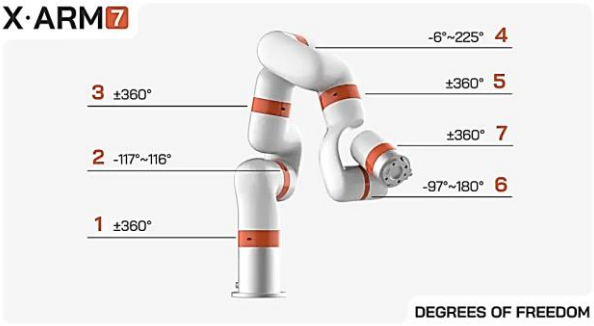

Fig. 1. xArm 7 manipulator with joints limits [21]

Within the proposed planning pipeline, the learning-based pre-filter rapidly discards sampled states that are likely to be in collision and forwards states that are likely collision-free to the planner. Occasional false-negative predictions are acceptable, since all candidate states are subsequently validated by an exact physics-based collision checker. Likewise, false-positive predictions discard valid states but do not compromise correctness, as they only reduce sampling efficiency.

For manipulators with kinematic redundancy, this trade-off is particularly favorable. Due to redundancy, multiple collision-free paths typically exist between the start and goal configurations. Therefore, rejecting a small subset of valid states does not prevent the planner from finding a feasible path. At the same time pre-filtering colliding states significantly reduces the number of expensive collision checks.

This leads to a conceptual distinction between conventional collision checking pipelines and the proposed pre-filtered approach. In the baseline approach, every sampled arm state is subjected to an exact collision check, making collision detection the primary computational bottleneck in sampling-based planning.

In contrast, the proposed pipeline introduces a lightweight learning-based pre-filter that operates on batches of sampled states. Batch sampling and batch neural inference are essential to achieving computational gains, as they amortize data preparation overhead and efficiently utilize parallel hardware. The pre-filter reshapes the sampling distribution by removing states that are likely to be in collision before exact validation.

1.2. State of the art

Existing research can be grouped into several main directions, which are analyzed below.

1) GPU-accelerated collision detection.

Early progress in accelerating collision queries was achieved through parallelization on graphics hardware. Pan and Manocha [5] introduced packetized BVH traversal for GPUs, reporting very high throughput of binary queries for large sets of robot states. Their approach demonstrated that parallel hardware can drastically reduce latency in collision checking. However, GPU-based methods still rely on explicit geometric models and provide only a binary in/out-of-collision response at the configuration level. They do not offer continuous severity measures (e.g., penetration depth) or link-wise resolution, which limits their integration as heuristics into higher-level motion planning. Despite these limitations, GPU-based acceleration remains a strong baseline for applications where hardware parallelism is available and binary decisions are sufficient.

2) Configuration-space proxies and learning-based classifiers.

A second research direction focuses on proxy models that approximate the collision boundary in configuration space. Das et al. introduced the Fastron framework [6], which uses online kernelized learning to approximate the feasible space and allows fast incremental updates

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

when the environment changes. This method has been shown to accelerate planning loops significantly. Later, Das and Yip [7] extended this work into a more comprehensive IEEE Transactions on Robotics study, analyzing proxy collision detectors as formal approximations with convergence properties and providing extensive evaluations across robotic benchmarks. These studies showed that learning-based proxies can achieve large speedups while maintaining safety guarantees, provided that conservative thresholds are applied. However, proxy methods generally return only binary collision labels and lack the ability to capture link-specific severity. This is a key limitation for manipulators where different links may contribute differently to collision risk.

3) Neural signed-distance field (SDF) approaches and differentiable proxies.

Recent research has explored continuous collision representations. DiffCo [8] proposed an auto-differentiable proxy that outputs distance-like measures and supports gradient-based optimization. This allows trajectory optimizers to receive smooth feedback signals rather than discrete labels, enabling gradient-based planners to converge faster. Similarly, iSDF [9] introduced a real-time neural reconstruction of signed distance fields from depth data, making it possible to query distance values for arbitrary points in space during planning. These approaches highlight the benefits of differentiability, but their primary focus is on optimization-based planning. They are less effective in scenarios requiring the high-throughput screening of tens of thousands of discrete candidate states, as occurs in sampling-based planning.

4) Implicit neural geometry.

Another recent trend is the use of implicit neural representations. Neural Collision Fields (NCF) [10] learn implicit occupancy functions over mesh primitives, reducing the need for explicit geometric contact queries. Neural Implicit Swept Volumes (NISV) [11] extend this to entire motion segments, predicting whether continuous motion intervals will intersect with obstacles. These methods are powerful for simulation, graphics, and physics applications where continuous reasoning is essential. However, their training cost is high, and their applicability to large-scale batched state filtering for motion planning remains limited. Importantly, these approaches typically operate at the level of primitives or trajectories, not individual manipulator links.

5) Real-time redundancy-aware collision avoidance.

With the rise of redundant and dual-arm manipulators in industrial and service robotics, new works target real-time collision avoidance in high-dimensional spaces. Sun et al. [12] proposed a method for real-time collision avoidance in dual-arm redundant robots operating in open environments. Their approach combines geometric reasoning with task-level redundancy resolution, ensuring safe collaboration in constrained workspaces. Zhang and Wang [13] present a redundancy-based motion-planning framework that explicitly enforces task constraints for 7-DoF manipulators, leveraging redundancy to improve connectivity yet still relying on geometric collision checks. Scoccia et al. [14] propose an online perturbation strategy of off-line generated trajectories for dynamically varying environments, combining potential-field planning with Bézier smoothing to maintain feasibility in real time. A recent survey by Zhang et al. [15] further explores motion planning for redundant space manipulators, identifying open challenges such as dynamic planning under uncertainty, multi-arm coordination, and efficiency constraints that call for scalable collision-checking front-ends. These studies show a clear trend towards dynamic, constraint-aware planning. However, they still lack a fast screening primitive for batched, per-link depth estimation, which is the specific gap this work addresses.

6) Classical motion planning foundations.

The foundations of collision-aware planning were established by classical methods that remain relevant today. Kavraki et al. [16] introduced probabilistic roadmaps (PRMs), which became a cornerstone for high-dimensional planning. Optimization-based planners such as CHOMP [17] and sequential convex optimization [18] advanced the integration of trajectory smoothness and collision constraints, while Quinlan & Khatib [19] developed the elastic bands method to bridge planning and control. Broad reviews such as Elbanhawi & Simic [20] provide systematic comparisons of sampling-based approaches. These classical methods typically rely on exact geometric collision checking, which remains a performance bottleneck in high-dimensional redundant manipulators.

Comparative analysis and identified gap

- GPU methods provide speed but are limited to binary outputs;
- Proxy models accelerate planning but lack per-link granularity;
- Differentiable SDF-based methods are useful for optimization but not efficient for batched screening;
- Implicit neural models reduce explicit contact computations but are costly to train and not per-link;
- Redundancy-aware approaches focus on task-level safety but still rely on geometric checkers;
- Classical methods laid the groundwork but suffer from scalability issues.

Across all these directions, there remains a research gap: methods that can efficiently predict per-link penetration depth, in large batches, conditioned on localized voxel evidence. This capability is particularly valuable for redundant manipulators, where link-wise risk estimation enables more nuanced filtering and prioritization

during motion planning.

The present work fills this gap by introducing a compact hybrid architecture (FC + KAN) that processes localized voxel depth patches together with link poses, delivering per-link penetration depth estimates. Unlike prior approaches, the proposed method explicitly targets the high-throughput pre-filtering stage, complementing rather than replacing exact physics-based collision checkers.

1.3. Objectives and tasks

The goal of this study is to accelerate collision checking for redundant robotic manipulators in mapped environments by designing a learning-based pre-filter that estimates per-link penetration depth, thereby reducing the number of expensive physics-based checks.

The objectives are as follows:

- to design a compact per-link input representation combining local voxel depth patches with link pose features;
- to develop and train a hybrid FC+KAN model for per-link penetration depth regression;
- to evaluate the proposed method against physics-based baselines in terms of regression accuracy, classification recall, and computational throughput;
- to analyze the practical implications for sampling-based motion planning pipelines.

In addition to computational efficiency, the proposed approach explicitly considers safety-related requirements. Since false-negative collision predictions may lead to unsafe robot behavior if used alone, the model is designed as a conservative pre-filter that prioritizes recall over precision. All candidate states that pass the pre-filter are subsequently validated using an exact physics-based collision checker, which remains the final safety gate in the planning pipeline.

The overall approach is to represent each candidate state with localized voxel evidence and pose features, predict per-link depths in batches, and use these predictions to filter out low-risk states before invoking exact collision queries.

The structure of the paper is as follows: the Section 2. Materials and Methods of research contain the following sub sections: 2.1. Data Representation and Input Formulation, 2.2. Network Architecture, 2.3. Training Data Generation, 2.4. Integration with Motion Planner, 2.5. Training Procedure, 2.6. Evaluation Metrics. Following sections are 3. Results and Discussion, and 4. Conclusions.

2. Materials and methods of research

This section details the proposed method, covering the data representation, the neural network architecture, the training data generation pipeline, and the heuristic for integrating the trained model into a sampling-based motion planner.

2.1. Data Representation and Input Formulation

To enable the network to make predictions based on local geometry, a specific input representation is formulated for each of the manipulator's links. The manipulator is modeled as a kinematic chain of 7 arm links plus a wrist-mounted tool, resulting in 8 distinct bodies for which collision depth is predicted.

For each of these 8 bodies, the input vector is constructed from three components:

1. Link Pose**:** The pose of a predefined reference point on the link is given relative to the robot's base frame. It is represented by its position and orientation, flattened into a 7-dimensional vector:

$$p_{link} = [x, y, z, q_x, q_y, q_z, q_w] \in \mathbb{R}^7, \quad (1)$$

where (x, y, z) – a link translation in robot's base coordinates;

(q_x, q_y, q_z, q_w) – a unit quaternion with a canonical representation ($q_w \geq 0$).

2. Voxel Grid Origin**:** The origin of the local voxel grid associated with the link, also expressed in the robot's base frame. This is represented by a 3-dimensional position vector:

$$p_{grid} = [x_g, y_g, z_g] \in \mathbb{R}^3, \quad (2)$$

where (x_g, y_g, z_g) - a translation of the local grid in a robot's base frame.

3. Local Voxel Grid**:** A small, axis-aligned cube of scalar values representing the pre-calculated signed penetration depth of the static environment around the link. The size of this grid, $S_i \in \{3, 5, 7\}$, is chosen heuristically based on the physical dimensions of the corresponding link i. The resolution of each voxel is 4 cm. The grid is flattened into a vector:

$$v_{grid} = [d_1, d_2, \ldots, d_{s_i^3}] \in \mathbb{R}^{s_i^3}, \quad (3)$$

where d_k – a penetration depth of the k-th voxel in flattened vector.

The values d_k are negative if no collision is present at that voxel's location and positive otherwise, representing the penetration depth.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

The complete input feature vector for a single arm state is formed by concatenating the representations for all 8 bodies, resulting in a total input dimension of 1854. The model's output is an 8-dimensional vector, where each component corresponds to the predicted penetration depth for one of the 8 bodies:

$$d^* = [d_1^*, d_2^*, \ldots, d_8^*] \in \mathbb{R}^8, \quad (4)$$

where d_k^* – a depth of collision of the *k*-th arm link with the environment, the robot's base, or other parts of the arm itself (self-collision).

In Figure 2 a visualization of a voxel grid calculated for an instrument installed on a robot wrist can be seen. Voxels that don't collide with a map are not displayed and voxels with a collision displayed as red cubes with side length the same as their collision depth.

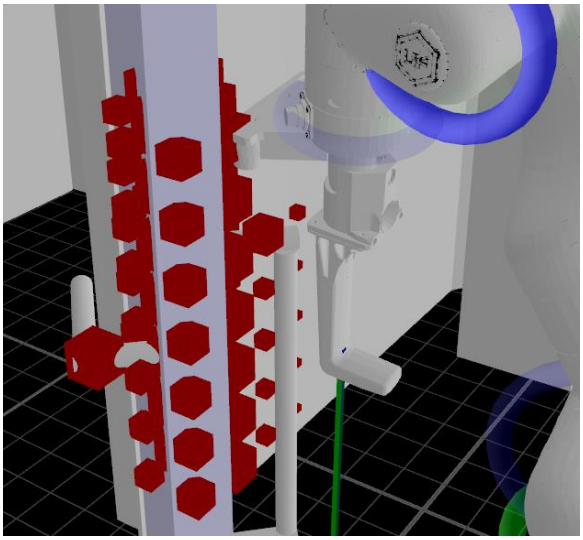

Fig. 2. Visualization of a voxel grid generated for an instrument link

2.2. Network Architecture

The proposed model is a hybrid architecture designed to first reduce the high-dimensional input into a compact representation and then process it through a Kolmogorov-Arnold Network (KAN) for the final regression task. The architecture consists of three main stages:

1. Projection Layer: The input vector $x \in \mathbb{R}^{1854}$ is first passed through a linear layer without bias to project it into a lower-dimensional space of 256 features.

$$h_1 = W_{proj} \cdot x, \quad (5)$$

where $W_{proj} \in \mathbb{R}^{256\times1854}$ - a projection layer.

2. Normalization and Activation**:** The projected features are normalized using Layer Normalization to stabilize training, followed by a GELU (Gaussian Error Linear Unit) activation function

$$h_2 = \mathrm{GELU}(\mathrm{LayerNorm}(h_1)). \quad (6)$$

3. KAN Block: The resulting 256-dimensional vector is processed by a multi-layer KAN. The KAN architecture is defined by the layer widths [256, 128, 64, 8], consisting of three learnable layers that regress the final 8-dimensional collision depth vector. The KAN implementation is based on the work presented in [4].

In Figure 3 a diagram of the network architecture can be seen.

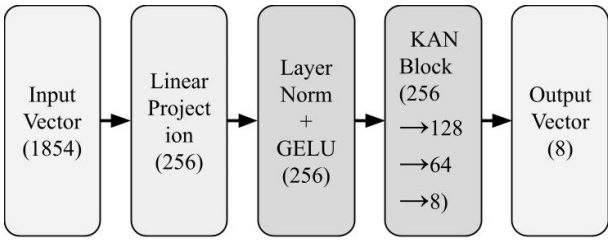

Fig. 3. Diagram of a network architecture

The choice of a Kolmogorov–Arnold Network (KAN) is motivated by its ability to approximate multivariate nonlinear functions through compositions of univariate spline functions, resulting in a compact parameterization and stable training behavior for regression tasks. In preliminary experiments, KAN-based models demonstrated comparable accuracy to deeper multilayer perceptrons with fewer parameters, which is beneficial for high-throughput inference.

While federated learning could in principle be considered for scenarios involving distributed robotic platforms with privacy-constrained local datasets, it was not required in this work. The proposed model is trained entirely in simulation, where collision labels can be generated at scale across diverse environments and configurations using an exact physics-based collision checker. Therefore, centralized offline training was both feasible and more efficient than federated or distributed learning for the considered setting.

2.3. Training Data Generation

A large-scale dataset was procedurally generated to train the network using code partially provided in Appendix A4. The process involves creating diverse collision scenarios by randomizing both the environment and the manipulator's configuration.

1. Environment Generation: A set of 4096 unique environments was generated. Each environment is populated with 17 primitive objects placed randomly within

the robot's workspace: 8 boxes with random dimensions and poses, 6 spheres with random radii and positions, and 3 thin walls with random positions and orientations.

2. Arm State Sampling: For each of the 4096 environments, 2048 random arm configurations (joint states) were uniformly sampled between the joint limits of the manipulator.

3. Ground Truth Calculation: For each sampled arm state within a given environment, the ground truth collision data was calculated using the Bullet physics library [2]. An efficient voxel grid system with memoization was employed. A wrapper class accepts requests with a batch of voxel indices. And after calculating the collision depth of each voxel the result is stored in a special map by voxel index. So, the next time stored collision depth returned. If a robot's base changed the position or map state changed, then the wrapper clears all the stored values. All scene objects (the environment and the robot) are inflated by 0.005 m. The exact penetration depth for each of the eight bodies is calculated and a 0.01 m offset (the doubled 0.005 m inflation margin) is subtracted from this depth, resulting in -0.01 m for collision-free states. This approach encourages the model to return a graded non-collision signal in the [-0.01, 0) range, providing more nuanced information for states that are close to a collision.

This process resulted in a dataset of approximately 8.4 million (4096 environments × 2048 states) unique input-output pairs.

2.4. Integration with Motion Planner

The trained model is intended to be used as a fast pre-filter and heuristic guide within a sampling-based planner. As an example the following algorithm outlines a proposed integration strategy with an RRT-Connect-like planner:

1. Batch Sampling: Generate a large batch of N candidate arm states (e.g., N=1024).

2. Batch Inference and Filtering: Perform a fast, batched inference pass with the trained network for all N states. Discard any states where the predicted collision depth d_i^* for any link i exceeds a predefined threshold (e.g., $d_i^* > 0$).

3. Path Scoring and Prioritization: For the remaining valid candidates, trace a path towards the goal, sample points along it, and use the network to predict a total collision score for the path. Sort the candidates based on this score, prioritizing those with paths that are "least in collision."

4. Exact Validation: Select the top K most promising candidates and pass them to the standard planner extension step, which uses the exact collision checker to ensure safety.

2.5. Training Procedure

The model was trained for 20 epochs. The dataset was split into training (80%) and testing (20%) sets at the environment file level. The training was performed using the following hyperparameters:

- Optimizer: Adam;
- Loss Function: Mean Squared Error (MSE), defined as:

$$L_{MSE} = \frac{1}{B \cdot L} \sum_{i=1}^{B} \sum_{j=1}^{L} (d_{ij} - d_{ij}^*)^2 , \qquad (7)$$

where B – the number of samples in a batch;

$L = 8$ – the number of links;

d_{ij} – the ground truth depth;

d_{ij}^* – the predicted depth.

- Initial Learning Rate: 0.002;
- Learning Rate Scheduler: StepLR;
- Batch Size: 1024.

The implementation uses PyTorch. Training and experiments were conducted on a system with a 12th Gen Intel(R) Core(TM) i7-12700H, 32GB of RAM, and a GeForce RTX 3050 Mobile GPU.

2.6. Evaluation Metrics

The model's performance was evaluated from two perspectives: as a regression model and as a binary classifier.

- Regression Accuracy: The primary metric is the final Mean Squared Error (MSE) between the actual value of the collision depth for each link and the result returned by the model on the test set.
- Classification Accuracy: Binary labels were generated by comparing the collision depth with a threshold of 0.0, i.e., binary classification answered the question of whether there is a link collision with the surrounding world in this case. A full confusion matrix – True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN) – was computed.

3. Results and Discussion

This section presents the quantitative results from the model evaluation after 20 epochs of training.

The model demonstrates strong performance as a regression tool, achieving a final Mean Squared Error of 0.000148 on the test set. The training progression showed stable convergence, as detailed in Figure 4.

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

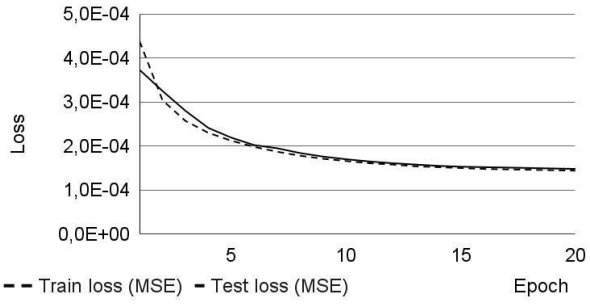

Fig. 4. Train and Test Loss (MSE) per Epoch

For its role as a pre-filter, the model's classification performance is crucial. Table 1 presents the per-link confusion matrix on the test set.

Table 1

Per-link classification confusion matrix on the test set

	Predicted Collision	Predicted No Collision
Actual Collision	TP: 4,242,931	FN: 319,092
Actual No Collision	FP: 667,116	TN: 8,205,740

Based on these results, the model achieves a per-link accuracy of 92.66%, a recall (sensitivity) of 93.01%, and a precision of 86.41%.

From a motion planning perspective, the reported per-link recall of 93.01% should be interpreted in the context of sampling-based algorithms. Since exact collision checking remains the final validation step, false-negative predictions do not result in unsafe trajectories but only reduce the effectiveness of early pruning. Similarly, false-positive predictions reduce the diversity of sampled states but do not affect probabilistic completeness, particularly for kinematically redundant manipulators where multiple feasible paths typically exist.

Performance benchmarks were conducted to compare the proposed method against direct collision checking with the Bullet physics engine. The total time for the proposed pipeline includes both the generation of local voxel grids (data preparation) and the neural network inference. The results for a batch of 8192 states are shown in Table 2.

The experimental results show that the proposed approach is effective. The low final MSE of 0.000148 (Figure 4) indicates that the model successfully learned the complex relationship between the robot's configuration, local geometry, and penetration depth.

From a practical standpoint, the classification metrics are more revealing. The model's high recall of 93.01% is particularly important; it means that about 93% of all actual collisions are correctly identified. A missed collision prediction (False Negative cases) is undesirable, as it reduces the effectiveness of early pruning. However, it does not compromise functional safety, since all candidate states are subsequently validated using an exact physics-based collision checker. The model's FP rate is also reasonably low, ensuring that the majority of collision-free states are correctly identified as such.

Table 2

Performance comparison for a single state, averaged over a large batch

	Batch Size	**Total Time Spent, ms**	**Time Per State, ms**	**Time Relative to Bullet**
Collision detection with Bullet	8192	1892	0.231	100%
Voxel grids generation	8192	319	0.039	16.9%
Neural network inference	8192	192	0.023	10.1%
Voxel grids generation + NN inference	8192	511	0.062	27.0%

Although Table 2 reports performance results for a batch size of 8192, additional profiling indicates that the relative speedup remains stable between 1024 and 8192 batch size. In this range both voxel grid generation with caching and neural network inference scale approximately linearly with batch size, whereas physics-based collision checking exhibits higher constant overhead. Therefore, the reported 3.7x speedup can be considered representative of typical operating conditions in sampling-based motion planning pipelines. At the same time with batch size lower than 1024 the relative speed-up drops, which indicates that this approach is most effective when applied to large batches of sampled states.

The proposed approach primarily targets different objectives than contemporary learning-based methods such as DiffCo [8] and Neural Collision Fields (NCF) [10], although all aim to accelerate collision reasoning. DiffCo provides a fully auto-differentiable collision proxy, making it highly suitable for gradient-based

trajectory optimization; its design goal is to supply stable gradients rather than to maximize the throughput of batched screening of independent configurations. NCF learns implicit collision representations at the triangle-primitive level, which is geared toward continuous contact handling in physics/graphics simulations rather than high-throughput per-configuration screening for robotic manipulators.

In contrast, the presented approach is optimized as a fast, batched pre-filter within sampling-based planners: it outputs per-link penetration-depth estimates (without exposing gradients), enabling both the filtering of invalid states and the prioritization of promising candidates via a nuanced collision score. It makes the approach complementary to optimization-focused or simulation-focused techniques, targeting a distinct stage of the planning pipeline. A direct head-to-head comparison is beyond the scope of this work due to differing problem formulations and evaluation metrics, instead, we report throughput and false-negative rates appropriate for pre-filter use.

The performance benchmarks in Table 2 quantify the primary motivation for this work. The complete proposed pipeline (data preparation + inference) requires only 0.062 ms per state, a 3.7-fold speedup compared to the 0.231 ms required by the exact checker. This shows great potential for accelerating sampling-based planners, which must evaluate tens of thousands of states.

A key factor in this performance is the use of an efficient, two-level system for generating voxel grids. While constructing these grids could be computationally intensive, the heavy lifting is done once per static environment by pre-calculating a global grid. The per-state cost is then reduced to a fast lookup operation. As a practical example, generating the input data for a batch of 8192 states required instead of 15,157,968 (8,192x1,854) only 69,349 unique contact tests with the physics engine due to caching by voxel index. This optimization is critical to the method's efficiency, making the data preparation step significantly faster than direct collision checking.

4. Conclusions

This paper demonstrated that learning-based per-link penetration depth prediction can significantly reduce collision-checking time for multi-joint robotic manipulators without compromising correctness. The proposed method reshapes the sampling distribution by acting as a statistical pre-filter, enabling the early rejection of collision-prone states while preserving functional safety through mandatory exact collision checking. Experimental results show a 3.7x speedup compared to direct physics-based collision checking, while maintaining a high per-link recall of 93.01%. Although the experiments focus on a kinematically redundant manipulator, the proposed pre-filtering approach is equally applicable to non-redundant multi-joint manipulators, as it does not rely on redundancy-specific properties.

The current approach has several limitations that define clear directions for future research. First, the model operates on a "snapshot" of the world, assuming the environment is static for the duration of a single planning query. Adaptation to dynamic obstacles moving during the planning cycle remains an open challenge. Second, and relatedly, this work assumes the manipulator's base is stationary relative to the environment within that same planning instance; extending the methodology to mobile manipulators, where the base and arm may move simultaneously, would introduce additional kinematic and state-representation complexity. Crucially, the final validation by an exact collision checker remains the mandatory safety gate; the False Negative rate, while low, must be carefully monitored in deployment, for instance, on a per-batch basis.

Future work will focus on two main directions. First, the model will be integrated into a full motion planning framework to benchmark the end-to-end reduction in planning time on standardized tasks. Second, the applicability of this approach to other manipulator geometries and a wider range of tools will be explored to assess its generalization capabilities.

Contributions of authors: conceptualization, methodology – **Andrii Medvid**; formulation of tasks, analysis – **Vitaliy Yakovyna**; development of model, software, verification – **Andrii Medvid**; analysis of results, visualization – **Andrii Medvid**; writing – original draft preparation – **Andrii Medvid**; writing – review and editing – **Vitaliy Yakovyna**.

Conflict of Interest

The authors declare that they have no conflict of interest in relation to this research, whether financial, personal, authorship or otherwise, that could affect the research and its results presented in this paper.

Financing

This study was conducted without financial support.

Data Availability

Data will be made available upon reasonable request.

Code Availability

The Python code for the model architecture, training, and evaluation is publicly available in a GitHub repository: https://github.com/amedvid/ArmLinksCollisionPredictor. The C++ code for data generation is not

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

fully available due to non-disclosure agreements. C++ code snippets added to the Appendix section.

Use of Artificial Intelligence

Generative AI tools (Gemini 2.5 Pro, ChatGPT 5) have been used for grammar checks, text polishing and additionally to assist in writing portions of the source code. The authors reviewed and edited all AI-generated content and took the responsibility for the final content of this publication.

Acknowledgments

The authors would like to express their deep gratitude to Somatic Holdings LTD, whose codebase greatly facilitated the development of the algorithm presented in this paper.

All the authors have read and agreed to the published version of this manuscript.

References

1. LaValle, S. M. *Rapidly-exploring random trees: A new tool for path planning*. Technical Report, 1998. Available at: https://lavalle.pl/papers/Lav98c.pdf (accessed September 2, 2025).

2. Coumans, E. *Bullet collision detection & physics library*. Available at: https://pybullet.org/Bullet/BulletFull/index.html (accessed October 2, 2025).

3. Medvid, A., & Yakovyna, V. Robot self collision prediction using Kolmogorov-Arnold networks. *Collection of scientific works of XXIII International scientific conference "Neural network technologies and their application – NNTA-2024"*, Kramatorsk-Vinnytsia-Ternopil, Ukraine, 2024, pp. 11-16. Available at: http://www.dgma.donetsk.ua/docs/news/2024/Збірник%20NNTA-2024.pdf (accessed September 2, 2025).

4. Liu, Z. *Efficient-KAN: An efficient pure-PyTorch implementation of Kolmogorov-Arnold Network (KAN)*. Available at: https://github.com/Blealtan/efficient-kan (accessed September 2, 2025).

5. Pan, J., & Manocha, D. GPU-Based Parallel Collision Detection for Real-Time Motion Planning. In: Hsu, D., Isler, V., Latombe, JC., Lin, M.C. (eds) Algorithmic Foundations of Robotics IX. *Springer Tracts in Advanced Robotics*, vol 68, 2010, Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-17452-0_13.

6. Das, N., Gupta, N., & Yip, M. Fastron: An online learning-based model and active learning strategy for proxy collision detection. *Proceedings of the 1st Conference on Robot Learning (CoRL 2017)*, Mountain View, CA, USA, 2017. pp. 496-504. Available at: https://proceedings.mlr.press/v78/das17a.html (accessed September 2, 2025).

7. Das, N., & Yip, M. Learning-Based Proxy Collision Detection for Robot Motion Planning Applications. *IEEE Transactions on Robotics*, 2020, vol. 36, iss. 4, pp. 1096–1114. DOI: 110.1109/TRO.2020.2974094.

8. Zhi, Y., Das, N., & Yip, M. DiffCo: Auto-differentiable proxy collision detection with multi-class labels for safety-aware trajectory optimization. *arXiv.org*, 2021. DOI: 10.48550/arxiv.2102.07413.

9. Ortiz, J., Clegg, A., Dong, J., Sucar, E., Novotny, D., Zollhoefer, M., & Mukadam, M. ISDF: Real-time neural signed distance fields for robot perception. *arXiv*.org, 2022. DOI: 10.48550/arxiv.2204.02296.

10. Zesch, R. S., Modi, V., Sueda, S., & Levin, D. I. W. Neural collision fields for triangle primitives. *SA'23: SIGGRAPH Asia 2023 Conference Papers*, Sydney, NSW, Australia, 2023, article no. 76, pp. 1–10. DOI: 10.1145/3610548.3618225.

11. Joho, D., Schwinn, J., & Safronov, K. Neural implicit swept volume models for fast collision detection. *arXiv.org*, 2024. DOI: 10.48550/arxiv.2402.15281.

12. Wu, Y., Jia, X., Li, T., & Liu, J. A real-time collision avoidance method for redundant dual-arm robots in an open operational environment. *Robotics and Computer-Integrated Manufacturing*, 2025, vol. 92, article no. 102894. DOI: 10.1016/j.rcim.2024.102894.

13. Zhang. Y., & Wang, H. Redundancy-Based Motion Planning with Task Constraints for Robot Manipulators. *Sensors*. 2025, vol. 25, iss. 6, article no. 1900. DOI: 10.3390/s25061900.

14. Scoccia, C., Palmieri G., Palpacelli,M. C., & Callegari, M. A Collision Avoidance Strategy for Redundant Manipulators in Dynamically Variable Environments: On-Line Perturbations of Off-Line Generated Trajectories. *Machines*, 2021, vol. 9, iss. 2, article no. 30. DOI: 10.3390/machines9020030.

15. Zhang, Z., Liu, X., Ning, M., Li, X., Liu, W., & Lu, Y. A review of motion planning for redundant space manipulators. *Science China Technological Sciences*, 2025, vol. 68, iss. 3, article no. 1310401. DOI: 10.1007/s11431-024-2841-y.

16. Kavraki, L. E., Svestka, P., Latombe, J. C., & Overmars, M. H. Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Transactions on Robotics and Automation*, 1996, vol. 12, iss. 4, pp. 566–580. DOI: 10.1109/70.508439.

17. Zucker, M., Ratliff, N., Dragan, A., Pivtoraiko, M., Klingensmith, M., Dellin, C., Bagnell, J.A., & Srinivasa, S. S. CHOMP: Covariant Hamiltonian Optimization for Motion Planning. *The International Journal of Robotics Research*, 2013, vol. 32, iss. 9–10, pp. 1164–1193. DOI: 10.1177/0278364913488805.

18. Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal,

I., Bradlow, H., Pan, J., & Abbeel, P. Motion Planning with Sequential Convex Optimization and Convex Collision Checking. *The International Journal of Robotics Research*, 2014, vol. 33, iss. 9, pp. 1251–1270. DOI: 10.1177/0278364914528132.

19. Quinlan, S., & Khatib, O. Elastic bands: connecting path planning and control. *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, USA, 1993, vol. 2, pp. 802–807. DOI: 10.1109/ROBOT.1993.291936.

20. Elbanhawi, M., & Simic, M. Sampling-Based Robot Motion Planning: A Review. *IEEE Access*, 2014, vol. 2, pp. 56–77. DOI: 10.1109/ACCESS.2014.2302442.

21. *UFACTORY XArm 7 Robotic Arms*. RobotShop Europe. (n.d.). Available at: https://eu.robotshop.com/collections/ufactory-xarm-7-robotic-arms (accessed September 2, 2025).

Appendix A: C++ Source Code Snippets

This appendix contains key C++ code snippets for reproducibility, illustrating the core components of the data generation pipeline. The code is simplified for clarity.

A.1 Voxel Grid Cache

To avoid redundant collision checks for the same voxel in space, a simple cache is used. It maps 3D integer coordinates to a pre-calculated collision depth value, using a 3D-to-1D index mapping for storage in a flat vector.

```
#include <vector>
#include <limits>
#include <stdexcept>

// Caches collision depth values for voxel indices to avoid redundant checks.
struct VoxelGridCache {
    static constexpr int MAX_INDEX = 75; // Defines the grid bounds, e.g., from -75 to 75.
    static constexpr int N = 2 * MAX_INDEX + 1; // Side length of the cache cube.
    static constexpr float EMPTY_VALUE = std::numeric_limits<float>::infinity();

    VoxelGridCache() : depthValues_(N * N * N, EMPTY_VALUE) {}

    // Checks if a depth value has already been computed for the given coordinates.
    [[nodiscard]] bool HasValue(const int x, const int y, const int z) const {
        return GetStoredDepth(x, y, z) != EMPTY_VALUE;
    }

    // Retrieves the cached depth value.
    [[nodiscard]] float GetStoredDepth(const int x, const int y, const int z) const {
        checkRange(x, y, z);
        return depthValues_[index(x, y, z)];
    }

    // Stores a new depth value in the cache.
    void SetStoredDepth(const int x, const int y, const int z, const float val) {
        checkRange(x, y, z);
        depthValues_[index(x, y, z)] = val;
    }

private:
    // Converts 3D voxel coordinates to a 1D vector index.
    static size_t index(int x, int y, int z) {
        const int ix = x + MAX_INDEX;
        const int iy = y + MAX_INDEX;
        const int iz = z + MAX_INDEX;
        return static_cast<size_t>((ix * N + iy) * N + iz);
    }

    // Ensures voxel coordinates are within the defined bounds.
    static void checkRange(int x, int y, int z) {
        if (x < -MAX_INDEX || x > MAX_INDEX || y < -MAX_INDEX || y > MAX_INDEX || z < -MAX_INDEX
|| z > MAX_INDEX) {
```

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

```
            throw std::out_of_range("VoxelGridCache: index out of range");
        }
    }
    std::vector<float> depthValues_;
};
```

A.2 Bullet Physics Contact Callback

A custom callback for Bullet's contactTest is used to find the maximum penetration depth. When a "probe" object collides with the environment, this callback records the deepest penetration found.

```
#include <btBulletDynamicsCommon.h>
// Custom callback to find the maximum penetration depth during a contact test.
class ProbeContactCallback : public btCollisionWorld::ContactResultCallback {
public:
    // Stores the maximum penetration found. Negative for separation, positive for penetration.
    btScalar maxPenetration = -0.01f;

    btScalar addSingleResult(
            btManifoldPoint& cp,
            const btCollisionObjectWrapper*, int, int,
            const btCollisionObjectWrapper*, int, int
    ) override {
        // Penetration distance is negative, so we invert it.
        btScalar penetration = -cp.getDistance();
        if (penetration > maxPenetration)
            maxPenetration = penetration;
        return 0; // Continue checking for deeper penetrations.
    }
};
```

A.3 Voxel Grid Population

This method is responsible for computing a local part of the global voxel grid. It iterates through the requested volume, checks the cache, and if a value is missing, performs a contactTest with Bullet to calculate it.

```
// (Inside VoxelCollisionGrid class)
// Computes a part of the grid, using the cache to avoid re-computation.
std::vector<std::vector<std::vector<float>>> VoxelCollisionGrid::ComputeGridPart(
        int startX, int startY, int startZ, int partSize) noexcept {

    std::vector<std::vector<std::vector<float>>> depthGrid(
        partSize, std::vector(partSize, std::vector(partSize, 0.0f)));

    for (int x = 0; x < partSize; ++x) {
        for (int y = 0; y < partSize; ++y) {
            for (int z = 0; z < partSize; ++z) {
                int currentX = startX + x;
                int currentY = startY + y;
                int currentZ = startZ + z;

                // 1. Check cache first.
                if (calculatedDepths_.HasValue(currentX, currentY, currentZ)) {
                    depthGrid[x][y][z] = calculatedDepths_.GetStoredDepth(currentX, currentY, cur-
rentZ);
                    continue;
                }

                // 2. If not in cache, compute using Bullet.
                auto voxelPos = WorldVoxelCenterIsometry(currentX, currentY, currentZ).Transla-
tion();
                btTransform tf;
                tf.setOrigin(btVector3(voxelPos.x(), voxelPos.y(), voxelPos.z()));
```

```
                probeObjectPtr_->setWorldTransform(tf);

                ProbeContactCallback callback;
                world_.contactTest(probeObjectPtr_, callback);
                contactTestCount_++; // For performance analysis.

                float depth = callback.maxPenetration;
                depthGrid[x][y][z] = depth;

                // 3. Store the newly computed value in the cache.
                calculatedDepths_.SetStoredDepth(currentX, currentY, currentZ, depth);
            }
        }
    }
    return depthGrid;
}
```

A.4 Random Scene Generation

The following functions are used to populate the simulation world with random obstacles for each of the 4096 environments.

```
// (Inside CollisionScene class)
// Populates the scene with a fixed set of random objects.
void CollisionScene::AddRandomObjects() noexcept {
    randomObjects_.push_back(AddRandomBox());
    randomObjects_.push_back(AddRandomBox());
    // ... (8 boxes total)
    randomObjects_.push_back(AddRandomSphere());
    // ... (6 spheres total)
    randomObjects_.push_back(AddRandomWall());
    // ... (3 walls total)
}

// Creates a single random wall and adds it to the Bullet world.
btCollisionObject* CollisionScene::AddRandomWall() const noexcept {
    constexpr float minWallThickness = 0.01f;
    constexpr float maxWallThickness = 0.12f;
    constexpr float wallSize = 1.6f;
    const float thickness = RandFloat(minWallThickness, maxWallThickness);
    btVector3 halfExtents(wallSize * 0.5f, thickness * 0.5f, wallSize * 0.5f);

    auto* shape = new btBoxShape(halfExtents);
    auto* obj = new btCollisionObject();
    obj->setCollisionShape(shape);
    obj->setCollisionFlags(btCollisionObject::CF_NO_CONTACT_RESPONSE);

    const float x = RandFloat(gridOffset_.x(), gridOffset_.x() + gridSize_.x());
    const float y = RandFloat(gridOffset_.y(), gridOffset_.y() + gridSize_.y());
    const float angleZ = RandFloat(0.0f, M_PI);

    btTransform tf;
    tf.setOrigin(btVector3(x, y, wallSize * 0.5f));
    tf.setRotation(btQuaternion(btVector3(0, 0, 1), angleZ));
    obj->setWorldTransform(tf);

    collisionWorld_->addCollisionObject(obj, int(CollisionGroup::RANDOM_MAP_OBJECT), -1);
    return obj;
}


// Creates a single random sphere and adds it to the Bullet world.
btCollisionObject* CollisionScene::AddRandomSphere() const noexcept {
```

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

```
    constexpr float minSphereRadius = 0.01f;
    constexpr float maxSphereRadius = 0.3f;
    const float radius = RandFloat(minSphereRadius, maxSphereRadius);
    auto* shape = new btSphereShape(radius);

    auto* obj = new btCollisionObject();
    obj->setCollisionShape(shape);
    obj->setCollisionFlags(btCollisionObject::CF_NO_CONTACT_RESPONSE);

    const float x = RandFloat(gridOffset_.x(), gridOffset_.x() + gridSize_.x());
    const float y = RandFloat(gridOffset_.y(), gridOffset_.y() + gridSize_.y());
    const float z = RandFloat(gridOffset_.z(), gridOffset_.z() + gridSize_.z());

    btTransform tf;
    tf.setOrigin(btVector3(x, y, z));
    tf.setRotation(btQuaternion::getIdentity());
    obj->setWorldTransform(tf);

    collisionWorld_->addCollisionObject(obj, int(CollisionGroup::RANDOM_MAP_OBJECT), -1);
    return obj;
}

// Creates a single random box and adds it to the Bullet world.
btCollisionObject* CollisionScene::AddRandomBox() const noexcept {
    constexpr float minBoxSize = 0.01f;
    constexpr float maxBoxSize = 0.5f;
    const float sx = RandFloat(minBoxSize, maxBoxSize);
    const float sy = RandFloat(minBoxSize, maxBoxSize);
    const float sz = RandFloat(minBoxSize, maxBoxSize);
    auto* shape = new btBoxShape(btVector3(sx / 2, sy / 2, sz / 2));

    auto* obj = new btCollisionObject();
    obj->setCollisionShape(shape);

    // Random position and orientation
    const float x = RandFloat(gridOffset_.x(), gridOffset_.x() + gridSize_.x());
    const float y = RandFloat(gridOffset_.y(), gridOffset_.y() + gridSize_.y());
    const float z = RandFloat(gridOffset_.z(), gridOffset_.z() + gridSize_.z());
    btQuaternion rot;
    rot.setEuler(RandFloat(0, M_PI), RandFloat(0, M_PI), RandFloat(0, M_PI));

    btTransform tf;
    tf.setOrigin(btVector3(x, y, z));
    tf.setRotation(rot);
    obj->setWorldTransform(tf);

    collisionWorld_->addCollisionObject(obj, int(CollisionGroup::RANDOM_MAP_OBJECT), -1);
    return obj;
}
```

A.5 Main Data Generation and Writing

This function orchestrates the entire data generation process for a single environment file. It samples arm states, calculates ground truth collisions and voxel grids, and writes all data to a binary file.

```
// (Inside CollisionNetTrainingGenerator class)
// Generates and writes training data for a number of arm states to a file.
void CollisionNetTrainingGenerator::WriteToFileArmCollisionsData(
        CollisionEngine& collisionEngine,
        model::ToolType toolType,
        const size_t numArmStates,
        const std::string& fileName
) const noexcept {
```

```
    // Define output file path
    auto fullOutFilePath = std::string("/path/to/output/data/") + ... + ".data";
    std::ofstream outFile(fullOutFilePath, std::ios::binary);

    // Creates a voxel grid cache object
       auto voxelGrid = collisionEngine.CreateVoxelGridForArm(worldType_, math::Isometry3d::Iden-
tity(), toolType);

    // Generate random arm states.
    auto states = arm::BoundingBoxStatesGenerator::GenerateStatesForRRISArmOnly(...);

    for (auto& state : states) {
        // 1. Calculate ground truth collision depths for all links.
        std::vector<float> collisionDepths(8, minCollisionDepth);
        auto collisions = collisionEngine.CollisionsList(state, worldType_);
        for (auto& collision : collisions) {
            ModifyCollisionDepths(collisionDepths, ...); // Update max depth per link
        }

        auto linksPositions = GetLinksWorldIsometries(collisionEngine.RobotModel(), state);

        // 2. For each link, write its pose and local voxel grid to the file.
        for (size_t linkIndex = 0; linkIndex < 8; linkIndex++) {
            // Write link pose (7 floats: x,y,z,qx,qy,qz,qw)
            auto linkPosFlat = Isometry3dToFlatNumbers(linksPositions[linkIndex]);
            outFile.write(reinterpret_cast<const char*>(linkPosFlat.data()), linkPosFlat.size() *
sizeof(float));

            // Write voxel grid origin (3 floats: x,y,z)
            int localGridSize = localVoxelGridSizes_[linkIndex];
            auto [x, y, z, voxelPosition] = GetVoxelOffsetAndPosition(linksPositions[linkIndex],
localGridSize);
            float pos[3] = {(float)voxelPosition.x(), (float)voxelPosition.y(), (float)voxelPosi-
tion.z()};
            outFile.write(reinterpret_cast<const char*>(pos), sizeof(pos));

            // Get local grid from the global grid (using the cache) and write it
            auto voxelGridPart = voxelGrid.ComputeGridPart(x, y, z, localGridSize);
            for (int i = 0; i < localGridSize; ++i) {
                for (int j = 0; j < localGridSize; ++j) {
                    outFile.write(reinterpret_cast<const  char*>(voxelGridPart[i][j].data()),  lo-
calGridSize * sizeof(float));
                }
            }
        }

        // 3. Write the ground truth collision depths vector (8 floats).
       outFile.write(reinterpret_cast<const        char*>(collisionDepths.data()),        colli-
sionDepths.size() * sizeof(float));
    }
    outFile.close();
}
```

Received 28.09.2025, Received in revised form 08.10.2025
Accepted date 17.11.2025, Published date 08.12.2025

ISSN 1814-4225 (print)
ISSN 2663-2012 (online)

ПОЛАНКОВЕ ПРОГНОЗУВАННЯ ГЛИБИНИ ПРОНИКНЕННЯ ДЛЯ НАДЛИШКОВИХ МАНІПУЛЯТОРІВ У РОБОЧИХ СЕРЕДОВИЩАХ

А. Я. Медвідь, В. С. Яковина

Предметом цього дослідження є перевірка колізій для надлишкових роботизованих маніпуляторів, що працюють у змінних середовищах, яка залишається суттєвим обчислювальним вузьким місцем у задачах планування руху. **Метою** дослідження є підвищення ефективності перевірки зіткнень для багатоланкових роботизованих маніпуляторів у вибіркових методах планування руху з одночасним збереженням функціональної безпеки. Це досягається шляхом розроблення та оцінювання методу на основі машинного навчання, який прогнозує глибину проникнення для кожної ланки та слугує статистичним попереднім фільтром, а не заміною точної перевірки зіткнень. Поставлені такі **завдання**: запропонувати нове подання вхідних даних, яке поєднує кінематичний стан маніпулятора з локалізованим геометричним контекстом, отриманим із середовища за допомогою воксельних ґрат; спроєктувати та реалізувати гібридну архітектуру нейронної мережі, що поєднує повнозв'язаний проєкційний шар із мережею Колмогорова–Арнольда (KAN); навчити модель на великому, процедурно згенерованому наборі даних із різноманітними сценаріями колізій; оцінити точність регресії, класифікаційні характеристики та прискорення обчислень порівняно з прямим фізичним перевірником колізій. Отримані такі **результати**: навчена модель демонструє високу регресійну точність із низьким середньоквадратичним відхиленням 0.000148 на тестовій вибірці; досягає обнадійливих результатів класифікації з відгуком 93.01 % для кожного лінка — це є важливим показником її придатності як попереднього фільтра, здатного відсіювати більшість небезпечних станів; за результатами продуктивності, для пакета з 8192 станів запропонований підхід (включно з підготовкою даних та інференсом) приблизно у 3.7 рази швидший за прямий фізичний перевірник колізій. **Висновки**. Наукова новизна отриманих результатів полягає у наступному: 1) запропоновано архітектуру нейронної мережі, що поєднує повнозв'язаний та Колмогорово–Арнольдівський шари для передбачення глибини колізій лінків надлишкового маніпулятора; 2) інтегровано кінематичні та воксельно-геометричні ознаки у єдине вхідне подання для точного оцінювання колізій. Запропонований метод ефективно виконує роль попереднього фільтра для планувальників на основі вибірки, зменшуючи кількість дорогих перевірок колізій і прискорюючи загальний процес планування руху.

Ключові слова: роботизовані маніпулятори; перевірка колізій; воксельні сітки; мережі Колмогорова-Арнольда.

Медвідь Андрій Ярославович – асп. каф. Систем Штучного Інтелекту, Національний Університет "Львівська політехніка", Львів, Україна.

Яковина Віталій Степанович – д-р техн. наук, проф., проф. каф. Систем Штучного Інтелекту, Національний Університет "Львівська політехніка", Львів, Україна.

Andrii Medvid – PhD Student of the Department of Artificial Intelligence Systems, Lviv Polytechnic National University, Lviv, Ukraine,
e-mail: andrii.y.medvid@lpnu.ua, ORCID: 0009-0001-4128-4973.

Vitaliy Yakovyna – Doctor of Technical Sciences, Professor, Professor of the Department of Artificial Intelligence Systems, Lviv Polytechnic National University, Lviv, Ukraine,
e-mail: vitaliy.s.yakovyna@lpnu.ua, ORCID: 0000-0003-0133-8591.