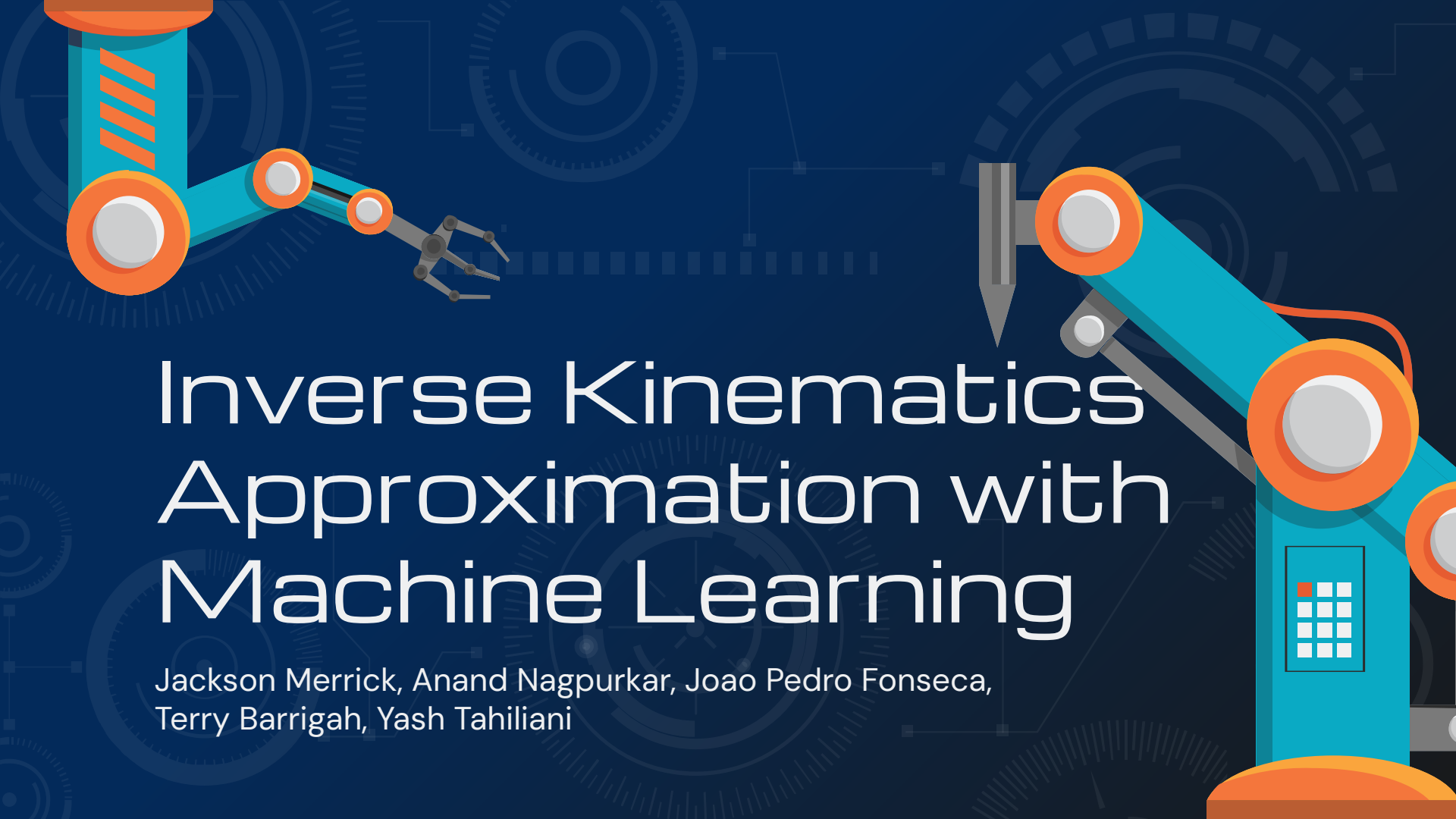


Inverse Kinematics Approximation with Machine Learning

Jackson Merrick, Anand Nagpurkar, Joao Pedro Fonseca,
Terry Barrigah, Yash Tahiliani

Background

Inverse Kinematics

Solve for joint angles (joint-space) from end-effector position (task-space)

Traditional Methods

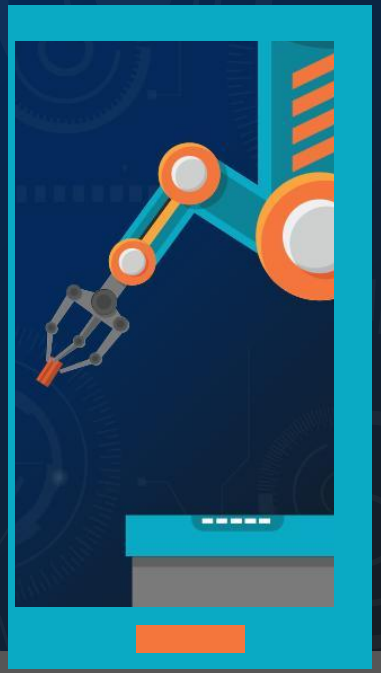
Traditional methods involve numerical solvers that are Jacobian-based and are iterative. Often are slow and provide one solution when there are multiple possibilities

Using AI/ML in IK

Neural network-based approaches to IK and reachability estimation can outperform conventional solvers in both speed and, with careful design, generalization.

Project Goals

- **Predict joint-space solutions for 4-DOF arms:** We aim to rapidly estimate feasible joint configurations for a desired end-effector pose using a machine learning regressor, bypassing slow iterative solvers. By mapping 3D position and yaw orientation to joint angles, our model enables real-time inverse kinematics suitable for online control.
- **Classify the feasibility and conditioning of postures:** We seek to identify, via classification models, whether a desired end-effector pose is reachable (i.e., there exists at least one valid solution) and whether the associated joint configuration is well-conditioned for reliable control.



Project Evolution

- **4-DOF IK Regression Pipeline:** Developed a robust supervised learning pipeline for 4-DOF arms, achieving sub-centimeter accuracy and efficient inference despite reduced complexity.
- **6-DOF Classification Approach:** Due to the high-dimensional, multi-modal nature of 6-DOF IK, we transitioned to classification models, predicting the feasibility and expected quality of IK solutions

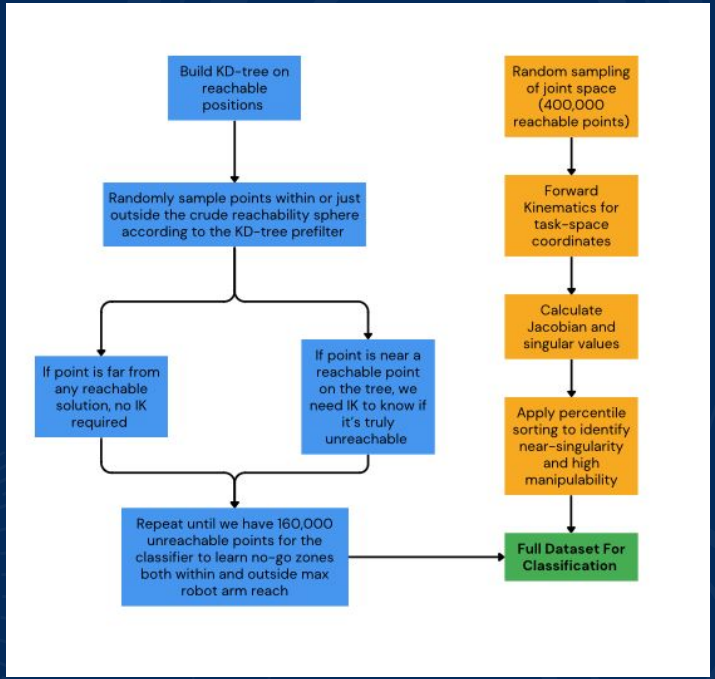
Problem Definition

6-DOF Feasibility & Conditioning Classifier

- Observed that directly learning inverse kinematics for a 6-DOF robot using regression was ineffective due to the complex and highly multimodal nature of the solution space.
- Implemented a classifier that categorises the quality of end-effector poses based on the inverse condition number of the Jacobian matrix. Labels we used are: unreachable, reachable but near-singular, reachable and adequately conditioned, and reachable with high manipulability.
- A k^{-1} value near zero indicates a singular configuration where certain directions of movement are lost, while a value near one implies high manipulability and stability.
- Manipulability refers to a scalar measure of how easily a robot's end-effector can move in different directions from a given posture.

Classification Data

- Joint space sampling for reachable points (N = 400,000)
- Reachable points classified on percentile basis of inverse kappa (bottom 20%, middle 60%, upper 20%)
- For unreachable points: KDTree filtering based on reachable dataset plus inverse kinematics fallback (N = 160,000; half inside arm sphere and half outside)



Classification Model

- **Model:** Multilayer perceptron implemented in PyTorch using torch.nn
- **Input:** 7D task-space vector [x, y, z, qx, qy, qz, qw].
- **Output:** 4 logits (one per class).
- **Hidden Layers:** Four nn.linear with hidden_size = 256 (optimized experimentally)
- **Activations:** nn.ReLU after each hidden layer introduces nonlinearity and prevents gradient shrinking
- **Final Layer:** Linear, feeding into CrossEntropyLoss()

```python
def __init__(self, hidden_size: int = HIDDEN_SIZE):
    super().__init__()
    self.net = nn.Sequential(
        nn.Linear(7, hidden_size),
        nn.ReLU(),

        # Linear + rectified lienar for nonlinearity
        nn.Linear(hidden_size, hidden_size),
        nn.ReLU(),

        nn.Linear(hidden_size, hidden_size),
        nn.ReLU(),

        nn.Linear(hidden_size, hidden_size),
        nn.ReLU(),

        nn.Linear(hidden_size, hidden_size),
        nn.ReLU(),

        # final logits → 4 classes
        nn.Linear(hidden_size, 4)
    )

def forward(self, x):
    """
    Args:
        x: Tensor of shape (B, 7)  where each row is [x,y,z,qx,qy,qz,qw]
    Returns:
        logits: Tensor of shape (B, 4)
    """
    return self.net(x)
```

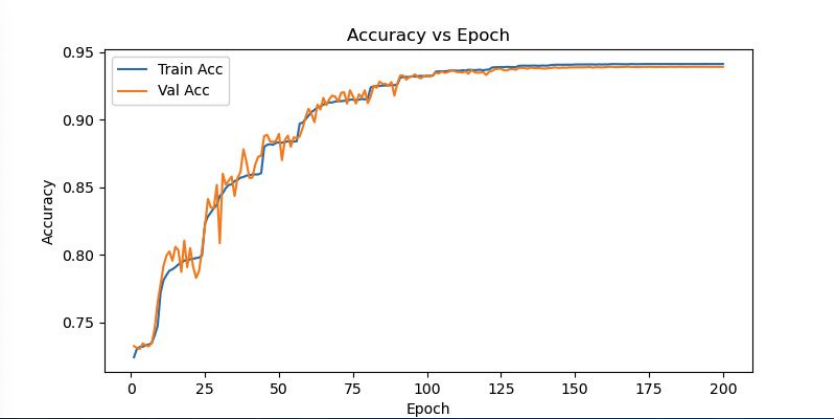
Classifier Training

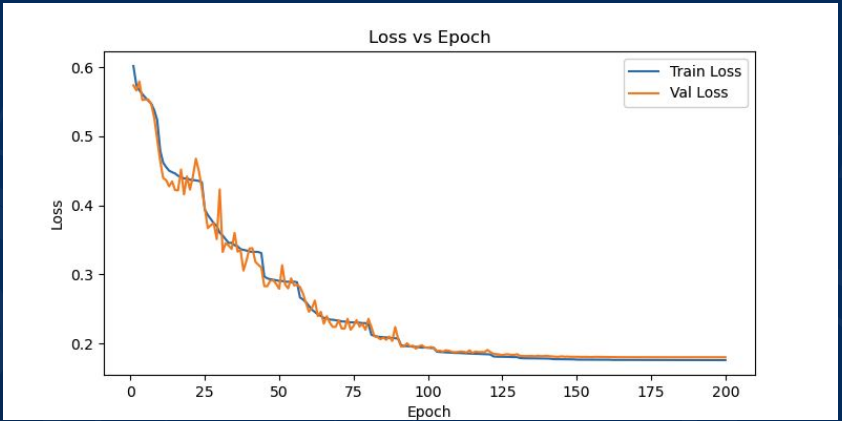
- **Optimizer:** torch.optim.Adam was chosen for its ability to adaptively tune learning rates. A weight decay of 1e-4 was added to reduce overfitting.
- **Loss Function:** torch.nn.CrossEntropyLoss() is favored for multiclass classification problems. Allows for optimization of confidence and not just accuracy
- **Learning Rate Scheduler:** ReduceLROnPlateau was used to reduce the learning rate on validation loss plateaus. Initial learning rate was 8.00e-3, final was 6.10e-8
- **Epochs:** Optimized experimentally, ran until significant learning rate decrement and final plateau (final training was 200 epochs).

Classifier Results

- Just shy of 95% accuracy
- Jagged val curves related to higher learning rate
- Stabilization within ~150 epochs

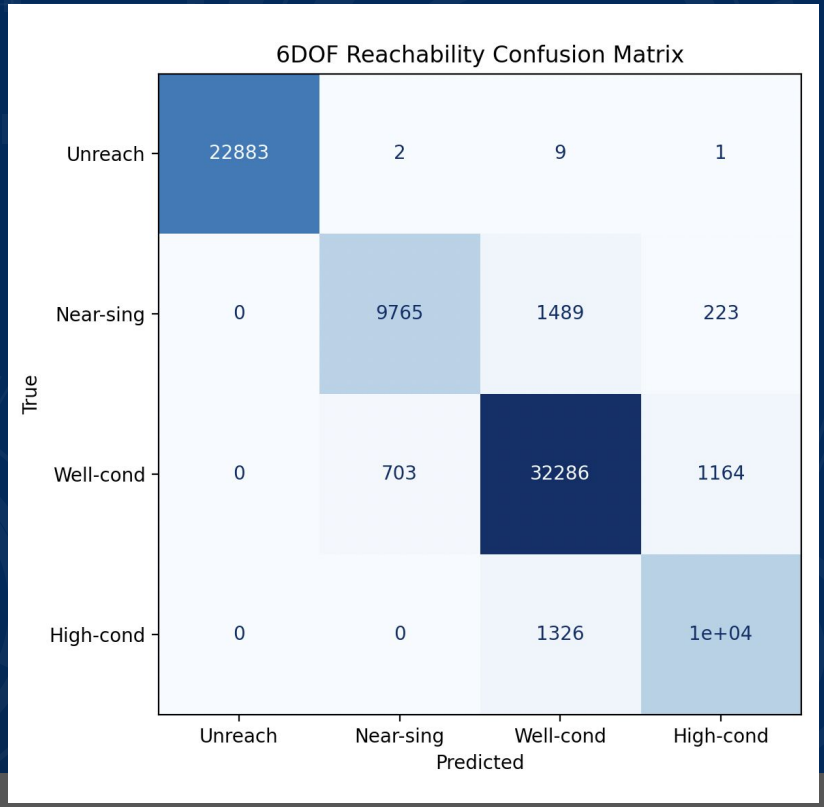
Training Accuracy	Training Loss (Cross Entropy)	Evaluation Accuracy	Evaluation Loss (Cross Entropy)	Final Test Accuracy
94.12%	0.1759	93.91%	0.1801	93.99%





Classifier Results

- High accuracy as binary reachability classifier (79,988/80,000)
- Percentile classification streamlines data generation but makes for blurry boundaries
- Overall shows promise for control/trajectory applications, could be coupled with IK fallback for 100% accuracy and ~95% less IK calls



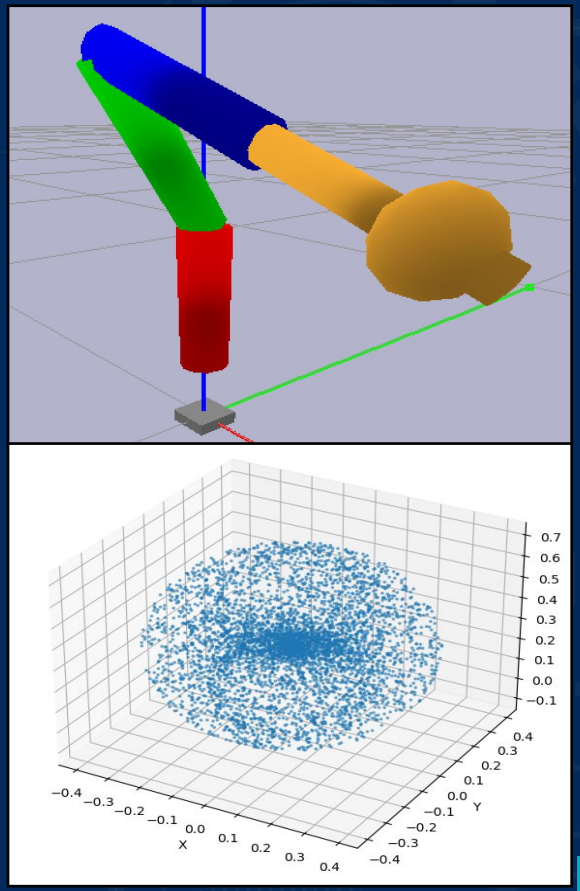
Problem Definition

4-DOF Direct IK Estimator (ML Model)

- We developed a supervised machine learning model that predicts a feasible and accurate joint configuration for a 4-DOF robotic arm, given a desired end-effector 3D pose
- To handle the multi-modal nature of the IK problem, we clustered the training data in joint space using KMeans, with each cluster representing a family of solutions.
- The 4-DOF robot was implemented in PyBullet with a custom URDF and synthetic datasets were generated using Latin Hypercube Sampling across each joint's limits.

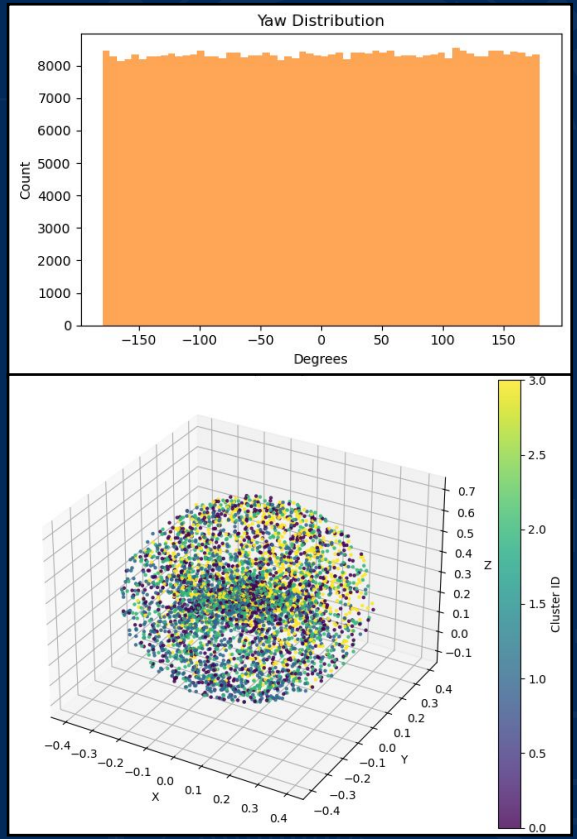
Model Data

- **Latin Hypercube Sampling:** 500,000 random samples generated for $[\theta_1, \theta_2, \theta_3, \theta_4]$ within joint limits ($-\pi$ to π).
- **Forward kinematics for each sample:** Computes corresponding end-effector position and orientation ($[x, y, z, q_x, q_y, qz, qw]$) using custom URDF and PyBullet.
- **Pose labeling:** Each data point paired with its originating joint angles, enabling direct mapping from (x, y, z, yaw) to $[\theta_1, \theta_2, \theta_3, \theta_4]$.
- **Cluster assignment:** K-Means clustering on joint angle space (n_clusters = 4) to group multiple IK solutions for a given pose (e.g., “elbow up”/“elbow down”).



Model Data

- **Train/test split:** Stratified sampling by cluster ensures balanced evaluation; typically 85% training, 15% test.
- **Preprocessing:** Standardization (mean, variance) of both input poses and joint angles to improve model convergence and generalization.
- **Visualization:** 3D scatter plots confirm uniform workspace coverage and highlight clustered structure of joint solutions.



IK Model

- **Model:** Multi-layer perceptron (MLP) implemented with scikit-learn's MLPRegressor.
- **Input:** 4D pose vector — [x, y, z, yaw] of the end-effector (yaw extracted from quaternion for simplicity).
- **Output:** 4 joint angles — $[\theta_1, \theta_2, \theta_3, \theta_4]$.
- **Hidden Layers:** Three fully-connected layers, each with 256 neurons (optimized experimentally).
- **Activations:** ReLU activation after each hidden layer to enable nonlinear mapping.

```
def __init__(self, base_regressor=None):
    """
    base_regressor: scikit-learn regressor instance
    (e.g., MLPRegressor, GradientBoostingRegressor).
    If None, defaults to MLPRegressor.
    """
    if base_regressor is None:
        self.base_regressor = MLPRegressor(
            hidden_layer_sizes=(256, 256, 256),
            max_iter=500,
            random_state=42
        )
    else:
        self.base_regressor = base_regressor
    self.cluster_models = {}  # cluster_label -> regressor
```

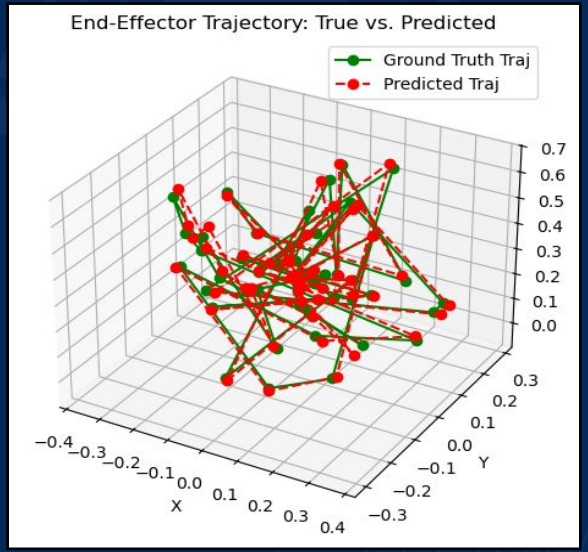
IK Model Training

Training Regime:

- **Optimizer:** Adam (default in scikit-learn), batch size = 1000, early stopping based on validation loss.
- **Data normalization:** StandardScaler on both input and output vectors.
- **Clustered Regression:** For multi-valued IK, K-Means clustering applied in joint space; separate MLP for each cluster improves solution diversity and accuracy.
- **Post-processing:** Best cluster solution selected using a forward kinematics (FK) consistency check (minimum pose error).

IK Model Results

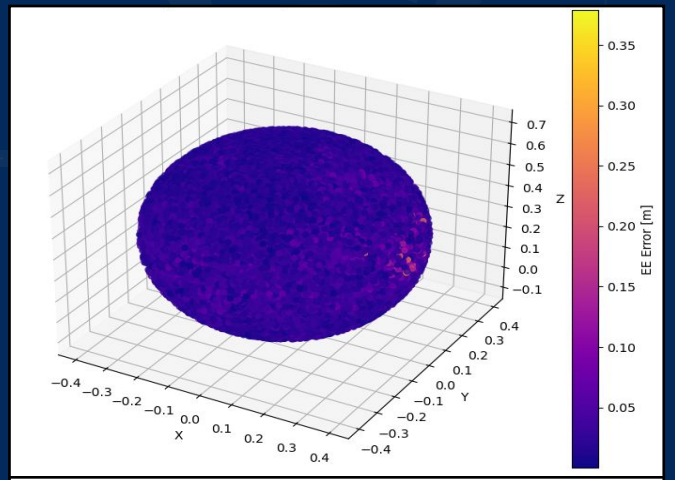
- Achieved sub-centimeter average end-effector error: Most predictions are within ~2 cm of ground truth position, within some applications' tolerances.
- MAE in joint space is higher, but physically unimportant: Multiple joint angle combinations can yield nearly identical end-effector positions, so end-effector (EE) error is a more meaningful metric for redundancy-rich arms.

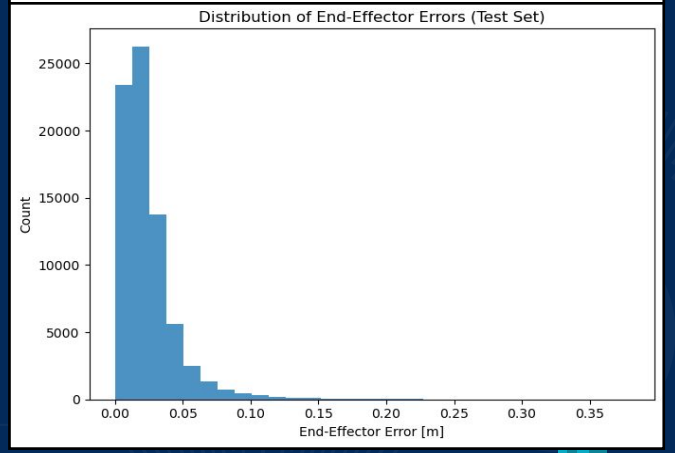


Method	Joint MAE (rad)	EE Error (m)	Total Time (s)	Time per sample (ms)
ML (Clustered, 4)	1.678	0.018	47.26	0.70
PyBullet IK	1.599	0.454	9.53	9.53

IK Model Results

- Workspace error heatmap shows low error across nearly all of the robot's workspace; outliers are rare and typically occur near the workspace boundaries or singularities.
- ML model is competitive with state-of-the-art IK solvers in both speed and accuracy for 4-DOF; inference is extremely fast (<1 ms per sample), compared to >10× slower analytical/numerical solvers.
- Visualization overlays and histogram confirm accurate and consistent predictions; the vast majority of errors cluster near zero with very few failures or large deviations.





Thank you!

References
[1] F. L. Tagliani, N. Pellegrini, and F. Aggogeri, “Machine learning sequential methodology for robot inverse kinematic modelling,” Applied Sciences, vol. 12, no. 19, p. 9417, Sep. 2022. doi:10.3390/app12199417
[2] M. N. Vu, F. Beck, M. Schwegel, C. Hartl-Nesic, A. Nguyen, and A. Kugi, “Machine learning-based framework for optimally solving the analytical inverse kinematics for redundant manipulators,” Mechatronics, vol. 89, p. 102970, 2023. doi:10.1016/j.mechatronics.2023.102970
[3] BioRob 4-DOF robot arm kinematic structure and table with DH parameters. Available: https://www.researchgate.net/figure/BioRob-4-DOF-robot-arm-kinematic-structure-and-table-with-DH-parameters_fig4_220850180
[4] Scikit-learn: StandardScaler documentation. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
[5] Scikit-learn: MLPRegressor documentation. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
[6] Robotics Unveiled: Velocity, Manipulability & Force Ellipsoids. Available: https://www.roboticsunveiled.com/robotics-velocity-manipulability-force-ellipsoids/
[7] R. F. Reinhart, Z. Shareef, and J. J. Steil, “Hybrid analytical and data-driven modeling for feed-forward robot control,” Sensors, vol. 17, no. 2, p. 311, Feb. 2017. doi:10.3390/s17020311
[8] J. Zeng et al., "Learning-based Inverse Kinematics for Redundant Robots: A Review," Journal of Computational Design and Engineering, vol. 11, no. 3, pp. 248-265, 2024. Available: https://academic.oup.com/jcde/article/11/3/248/7684300