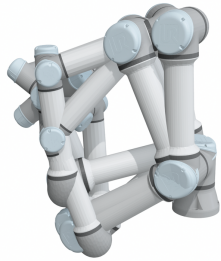
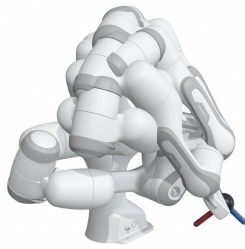


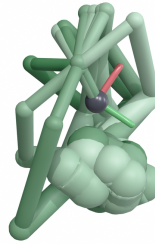
MorphIK: Morphology-Conditioned Neural Inverse Kinematics for Unknown Robots

Lennart Clasmeier¹Jan-Gerrit Habekost¹Cornelius Weber¹Stefan Wermter¹

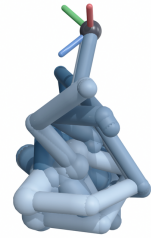
UR5e 6-DoF



Franka Panda 7-DoF



robot 002567 7-DoF



robot 000793 8-DoF

Fig. 1: Our neural inverse kinematics model generates a large variety of solutions for a single pose on real-world and procedurally generated arms, unseen during training

Abstract—Neural models can learn to generate various solutions to the inverse kinematics problem from data, but are usually limited to a single robot. We present MorphIK, a flow-matching model that solves inverse kinematics for revolute-joint-based kinematic chains it has never seen during training. The model uses a transformer architecture to encode the robot’s morphology along with the target pose. This encoding then conditions a flow-matching head that generates poses from noise. Trained on purely synthetic data from procedurally generated robots, the model reaches a precision of about 5 cm on unseen real-world robots with 6 to 9 Degrees of Freedom. For higher precision, the model serves as an excellent Prior for further optimization algorithms, reducing error to less than 1 cm after a single step of Damped Least Squares optimization and to sub-1 mm error after 3 steps in most cases. Building on flow matching’s generative capabilities to produce highly diverse outputs, our model can efficiently sample the robot’s null space, providing a wide variety of configurations for the same pose. Thus, overall MorphIK allows learning and generalizing neural inverse kinematics for a multitude of known and unknown robots.

Index Terms—robotics, inverse kinematics, neural networks

I. INTRODUCTION

Although inverse kinematics is ubiquitous in robotics, it remains an open challenge for redundant kinematic chains, with many approaches in parallel for different use cases.

The authors gratefully acknowledge funding from Horizon Europe under the MSCA grant agreement No 101226624 (GREET) and support from the German Research Foundation DFG under project LUMO (No 551629603).

¹ L. Clasmeier, J.-G. Habekost, C. Weber and S. Wermter are with the Knowledge Technology Group, Department of Informatics, University of Hamburg, Hamburg, Germany, e-mail: {lennart.clasmeier, jan-gerrit.habekost, stefan.wermter}@uni-hamburg.de

One family of approaches that has gained traction in recent years uses neural methods to learn an inverse kinematics model from data, rather than treating it as an optimization problem or seeking a closed-form solution. This approach offers multiple advantages over traditional methods, including runtime, redundancy, and singularity handling. Since neural inference can run batched on a GPU, many poses can be solved at once, enabling parallel exploration of the null space. Modern generative approaches like normalizing flows [1], GANs [2], diffusion [3], or flow matching [4] model one-to-many solutions and therefore are a natural fit for the multimodality of Inverse Kinematics for redundant manipulators. Learned models also do not rely on the kinematic chain’s Jacobian, so singular configurations become less severe.

While these advantages are promising, the biggest limitation of neural IK solvers is that they are usually single-robot models trained to solve the IK problem for exactly one kinematic chain. This results in costly data collection and retraining, a real hurdle for integrating and adapting these approaches. We propose MorphIK, a transformer-based neural multi-robot IK solver that generalizes to all kinds of 6- to 9-DoF unseen revolute serial manipulators. Our model requires only a robot’s URDF, from which we derive a canonical representation and tokenize it as the model’s geometry input.

A Transformer backbone combined with a flow-matching decoder generates IK solutions for a given pose target. Because the robot is an input rather than a fixed assumption, training requires not one robot but many. Thus, we train on a multitude of randomly generated robots covering a wide range of morphologies. We generate training data in a

fully self-supervised way by sampling configurations from the robot’s joint spaces and applying forward kinematics to obtain the corresponding poses. We store only robot descriptions and randomly generate poses on the GPU during training. Although the model has never seen a real robot during training, we show that it generalizes to a wide range of real-world manipulators, achieving less than 1 cm error after a single step and sub-millimeter accuracy after only 3 steps of post-inference polishing with DLS.

II. RELATED WORK

Many different IK algorithms have been proposed over the years. Traditional solvers like KDL [5] or Trac-IK [6] use the Jacobian of the kinematic chain to solve the problem, which can cause issues near singularities where the Jacobian determinant is 0 and velocities become infinite. Although damping and singularity avoidance can address these issues, they still create real limitations that must be worked around.

Genetic algorithms do not rely on the Jacobian, which makes singularities less of an issue for algorithms like Bio-IK [7]. Additionally, this class of algorithms can adapt to arbitrary constraints added to the optimization objective, but at the cost of higher computational overhead.

Neural Methods try to overcome this issue by learning to solve Inverse Kinematics from data. The main advantage is that inference on modern GPUs is extremely fast and parallel, and generative architectures can model multimodal distributions, allowing effective exploration of the robot’s null space. Further, different loss functions can adapt the model behavior to the desired behavior, and training data can be generated by calculating forward kinematics on random joint configurations.

IK Flow [1] uses normalizing flows to train a generative IK network to an accuracy of 1 cm and 2° of error. Additionally, it serves as a seed model for more effective null space sampling in Trac-IK. CycleIK [2] trains both a GAN and an MLP model for IK by utilizing differentiable forward kinematics to calculate position and orientation loss in Cartesian space. Generative Graphical Inverse Kinematics [8] trains a Graph Neural Network to solve the Inverse Kinematics problem by means of completing a partial graph of distances. It can incorporate multiple robots in the same model through a distance-geometric graph representation of manipulators using a set of fixed points. It also trains a model on a randomly generated dataset of robots, then evaluates it on real-world robotic manipulators with error ranges similar to our approach. IKDiffuser [3] generalizes neural IK to kinematic trees with multiple end effectors. By tokenizing the end effector together with a target, a single model can solve IK for multiple chains of the same robot. While the model provides a single formula applicable to all kinds of morphologies, such as hand kinematic trees, it remains a single-robot model that must be trained individually for each robot. MimicIK [4] uses flow matching to train a neural IK solver for teleoperation, focusing on trajectory smoothness and real-time performance from teleoperation data. We extend the flow matching approach to a larger

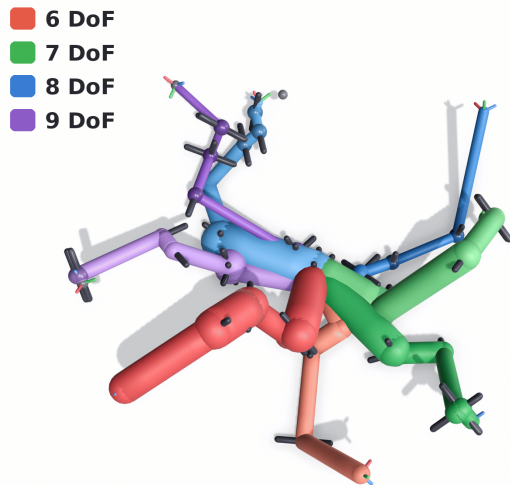


Fig. 2: Example robots from the procedural robot generator

conditioning vector, which includes a robot representation, turning it into a multi-robot model.

III. APPROACH

Our approach uses a tokenized canonical robot representation as conditioning for a flow matching model that predicts robot configurations. Since this conditioning requires a large variety of robots to generalize, we created a dataset of procedurally generated robots for training. We add a Cartesian auxiliary loss to the flow matching training objective, which speeds up and stabilizes training. Finally, we evaluate the model on a set of unseen real-world robots.

A. Canonical Robot Representation

Our approach generalizes neural IK to unseen robots by using procedurally generated robots as data generators. The generator samples random revolute kinematic chains and stores them in a canonical form. The canonical form removes redundancies from the robot representation, making robot morphology representations easier to learn because identical robots have identical representations. The robot is described as a chain of joints, each defined by a rotation axis in space through an axis point and a unit vector for the axis direction. Since we can choose the axis point freely by sliding along the axis, we canonicalize it as the point closest to the previous axis point in the chain. The first joint defines the origin: its axis becomes the $+z$ axis, and the axis point is placed where that axis comes closest to the next axis that is not parallel to it. To remove the remaining rotational freedom around the z -axis, we rotate about z until the first link that leaves the axis lies in the xz plane. Each remaining axis direction is fixed by requiring its first non-negligible component to be positive; flipping an axis mirrors that joint’s rotation, so its limits are negated and swapped. One freedom survives both steps: reversing the first joint’s axis describes the same robot. By construction, we rotate

that axis onto +z before we can decide its direction. We therefore canonicalize both variants and use the parameter vector with the smaller value at the first differing entry as a tie-breaker. A rigid tool transform describing the end-effector pose completes the robot description. Finally, we normalize the total chain length, including the tool segment, to one to create scale invariance. The result is a product-of-exponentials description: each joint is a screw axis expressed in one common frame, and forward kinematics is the product of the corresponding matrix exponentials [11].

Because all joints share one reference frame rather than the relative frames of the URDF or DH conventions, the resulting geometry tokens are directly comparable across joints, robots, and degrees of freedom, and the model never has to reconstruct the kinematic chain to relate them.

B. Dataset

We sample joint links from a Dirichlet distribution ($\alpha = 2$), with 35% of the links set to zero, which makes the adjacent axes intersect and creates pseudo-spherical joints. The rotational axes are sampled in a structured way: 25% of the joints are parallel to the previous joint, 55% orthogonal to it, and 20% uniformly on the sphere. This mix creates a highly varied set of synthetic kinematic chains to help with generalization, while introducing some structure resembling real-world robots.

For 7-, 8-, and 9-DoF chains, roughly 30% of the robots terminate in a compact two-joint wrist rather than distributing joints evenly along the whole chain. This layout is inspired by real-world scenarios, in which a robotic hand with its own wrist is attached to a cobot.

Joint limits are symmetric around zero, with the half-range drawn uniformly from $[0.9, \pi]$ for arm joints and $[0.15, 0.9]$ for wrist joints, since wrists are usually more limited in their movement. Placing the 0 value at the center of the range is a convention rather than a property of the mechanism, and the URDF importer re-centers every real arm the same way. Tool orientation is sampled uniformly from $SO(3)$ as a unit quaternion. The Capsule radii are sampled with a slight taper towards the end of the chain.

We sample 100,000 robots per DoF class, leading to a dataset of 400,000 robots, holding out 200 robots per class for evaluation. During training, we generate data online from this fixed set of robots, providing an unlimited supply of training samples. First, we sample joint values from the bounds, then check configurations for self-collision using a

simple capsule model. Finally, a GPU-accelerated, highly parallelized FK solver computes the forward kinematics, producing the final training triplet of robot description, target pose, and joint configuration. To compare the model’s ability to generalize to real hardware, we also evaluate it on a set of 6- and 7-DoF robotic arms from the robot_descriptions [12] Python package as real-world robot geometries. We further generated realistic 8- and 9-DoF chains by attaching the Shadow Hand, also available in this package and featuring a 3-DoF wrist, to the 6- and 7-DoF robots and deactivating its redundant wrist rotation joint. This yields 44 real robot geometries, in addition to the 800 held-out synthetic robots from the robot generator.

Table I shows the recorded metrics of the procedurally generated robot multitude compared to the evaluated 6-, 7-, 8-, and 9-DoF real-world kinematic chains. Our robots match the real arms in link length (about 35 cm), but their joint ranges are distributed differently, and their kinematic properties differ more substantially. While real manipulators often have large joint ranges that allow near-full motion, with some very limited joints like wrists, we sampled the synthetic joint ranges uniformly. We report manipulability [9] as a measure of how far a configuration sits from a singularity. It is the volume of the velocity ellipsoid at a single configuration, so larger values mean more end-effector motion is available per unit of joint motion. Since it is defined pointwise, we report the median over 32 randomly sampled configurations per robot. We also report the condition number of the Jacobian [10], the ratio of the ellipsoid’s longest to shortest semi-axis, which describes the shape rather than the size of the same ellipsoid; smaller is better, with 1 denoting an arm equally capable in every direction.

On both measures, the generated robots are significantly worse than the hand-crafted ones, and consistently so across DoF classes: their manipulability is roughly five times lower and their condition numbers roughly twice as high. In practice, this means the generated arms spend far more of their configuration space near singularities, and their motion capability is much more unevenly distributed across directions. Both properties affect the inverse kinematics directly, since a Jacobian-based solver must invert exactly this matrix at every iteration: a condition number of 72 rather than 32 means errors are amplified more than twice as much in the linear solve that the polishing step performs. The generated robots are therefore not simply unfamiliar but numerically

TABLE I: Dataset metrics for procedurally generated and real-world robots at median, [5th percentile, and 95th percentile]

	robots	joint range (rad)	longest link (m)	manipulability $\times 10^{-3}$ [9]	condition number [10]	3 parallel	3 meeting
6-DoF	99800	4.05 [2.03, 6.06]	0.37 [0.26, 0.56]	1.84 [0.37, 7.00]	71.81 [33.76, 171.44]	5%	38%
7-DoF	99800	3.90 [1.49, 6.05]	0.36 [0.25, 0.56]	6.34 [1.16, 21.85]	42.92 [22.26, 122.00]	9%	64%
8-DoF	99800	3.91 [1.59, 6.04]	0.33 [0.23, 0.53]	13.15 [2.96, 37.93]	32.65 [19.31, 83.28]	12%	76%
9-DoF	99800	3.92 [1.71, 6.05]	0.31 [0.21, 0.49]	22.18 [6.15, 56.76]	27.95 [18.08, 61.16]	13%	85%
real 6-DoF	14	6.28 [2.89, 6.28]	0.40 [0.31, 0.47]	8.92 [3.78, 19.44]	31.72 [20.04, 59.16]	57%	29%
real 7-DoF	8	5.87 [3.64, 6.28]	0.38 [0.35, 0.45]	30.22 [23.62, 74.12]	19.17 [15.50, 23.04]	12%	75%
real 8-DoF	14	5.04 [0.70, 6.28]	0.34 [0.28, 0.48]	53.24 [37.34, 123.59]	15.21 [11.95, 26.66]	54%	62%
real 9-DoF	8	5.75 [0.70, 6.28]	0.34 [0.28, 0.44]	135.29 [59.27, 210.97]	11.94 [10.23, 14.23]	14%	86%

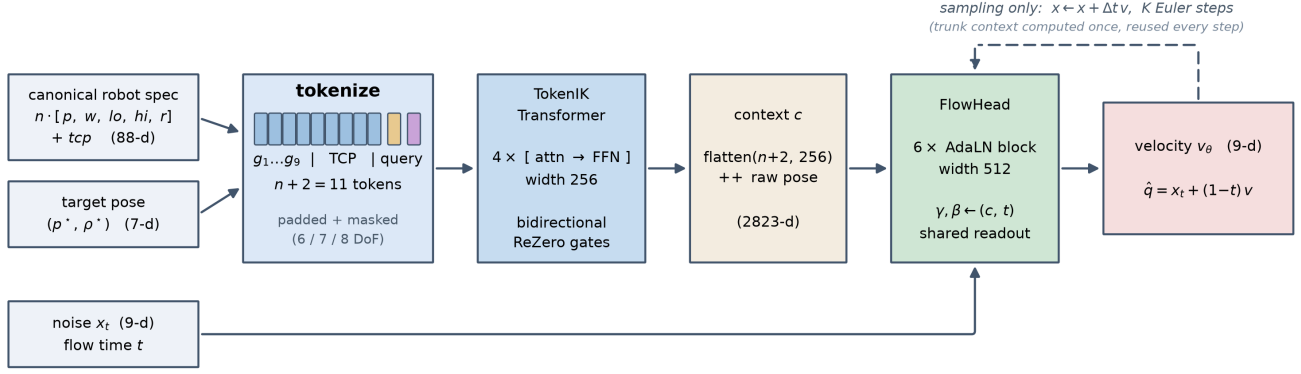


Fig. 3: Robot Geometry Tokens and Target pose are tokenized and used as input to the Transformer backbone. The model output is then flattened and used to condition the flow Matching Head. The flow head takes noise and flow time t as input and generates the velocity vector v_θ . At inference, this vector is applied to x_t and fed back into the flow head to create the next step.

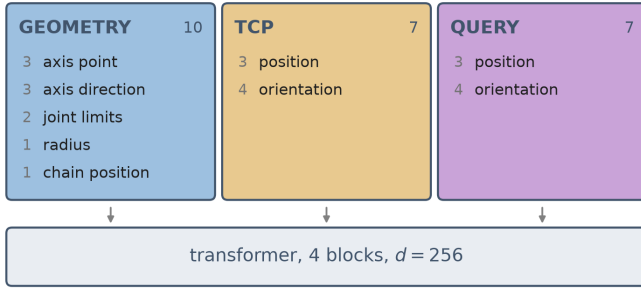


Fig. 4: Token layout of the MorphIK transformer backbone. The Geometry tokens contain the robot’s morphological information, with one token per joint. A single TCP token represents the end-effector pose at joint configuration 0, and a Query token defines the IK Pose target.

harder to solve than the real arms it is evaluated on.

Another common evaluation metric for robotic manipulators is the Pieper criterion [13], which defines whether a 6-DoF kinematic chain has a closed-form solution. A closed-form solution exists either if three consecutive joints are parallel, or if three consecutive axes meet in a single point, forming a spherical wrist. While 86% of the real 6-DoF arms satisfy at least one of the two Pieper criteria, only 42% of the generated 6-DoF chains do. The gap is entirely in the parallel case: 57% of real 6-DoF arms carry three consecutive parallel axes against 5% of ours, whereas on the intersecting criterion the generated chains slightly exceed the real ones (38% against 29%). The share rises with DoF for both populations because more joints yield more consecutive triples that may satisfy either criterion.

C. Model Architecture

The neural architecture shown in Fig. 3 uses a transformer trunk to encode the robot geometry and task, and a flow-matching head to generate the robot configuration. Similar

to the MetaMorph architecture [14], the encoder takes a sequence of geometry tokens containing one token per joint and a tool center point (TCP) token, plus a pose token that encodes the target end-effector pose as shown in Fig 4. While MetaMorph only handles robots from its own recipe, our canonical robot representation ensures that all kinds of robots can be encoded directly from URDF into the same space. The encoder generates the encoding with multiple rounds of multi-headed self-attention in a traditional encoder-only transformer architecture. The encoder output of $n + 2$ token embeddings is flattened into a conditioning vector for the flow matching head [15].

The flow matching head is an MLP with AdaLN [16] conditioning. The flow matching head takes noise as input and acts as a velocity field that integrates the noise over multiple passes. The AdaLN layer takes the task encoding from the transformer backbone as input. All robot architecture token sequences are padded and masked for the maximum 9-DoF sequence length, similar to the approach of π_0 [17].

D. Losses

The model is trained on a combination of flow matching loss and forward kinematic loss. Equation 2 shows the standard flow matching loss, with x_0 being a random vector, and q the joint configuration. v^* is the target velocity from the noise vector x_0 to the goal configuration q , and $\mathcal{L}_{\text{FM}}$ is the MSE between the predicted velocity and the target velocity v^*

$$x_t = (1 - t)x_0 + tq, \quad v^* = q - x_0 \quad (1)$$

$$\mathcal{L}_{\text{FM}} = \|v_\theta(x_t, t, c) - v^*\|^2 \quad (2)$$

Habekost et al. [2] proposed a cycle loss enabled by differentiable forward kinematics to compute position and orientation loss in Cartesian space, providing a strong training signal for IK and allowing deterministic networks to

commit to a single solution rather than averaging over multiple configurations for the same pose. We bring this approach into the domain of flow matching as an auxiliary loss that heavily stabilizes the training. Normal diffusion approaches struggle with this kind of loss because it requires an actual final output q . Since the learned vector field is trained without full inference, only relatively noisy estimates are available [18].

The flow matching model’s velocity field is mostly straight, so we can estimate the trajectory endpoint in a single step by following the predicted velocity directly to the end, yielding an estimate $\hat{q}$ of the output configuration. We then use this estimated output configuration to compute the estimated end-effector pose $(\hat{p}, \hat{q})$ via differentiable forward kinematics FK and, from that, the Cartesian position and orientation losses $\mathcal{L}_{\text{ori}}$ and $\mathcal{L}_{\text{pos}}$.

$$\hat{q} = x_t + (1 - t) v_\theta(x_t, t, c) \quad (3)$$

$$(\hat{p}, \hat{o}) = \text{FK}(\hat{q}) \quad (4)$$

$$\mathcal{L}_{\text{pos}} = \text{smooth}_{L_1}(\hat{p}, p^*) \quad (5)$$

$$\mathcal{L}_{\text{ori}} = \min\{\text{smooth}_{L_1}(\hat{o}, o^*), \text{smooth}_{L_1}(-\hat{o}, o^*)\} \quad (6)$$

Since the estimated joint position is only useful at the end of integration, and predictions tend to be poor at the beginning of the flow process, we time-gate the forward kinematic losses to the second half of the flow process. Additionally, the losses are scaled by t^2 which results in the time gate factor $\gamma(t)$

$$\gamma(t) = \begin{cases} t^2, & t \geq \frac{1}{2} \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

The final loss $\mathcal{L}$ is the expected value of the weighted sum of the three loss terms.

$$\mathcal{L} = \mathbb{E}\left[\lambda_{\text{FM}} \mathcal{L}_{\text{FM}} + \lambda_{\text{FK}} \gamma(t) (\omega_p \mathcal{L}_{\text{pos}} + \omega_o \mathcal{L}_{\text{ori}})\right] \quad (8)$$

The transformer encoder and the flow matching decoder are trained end-to-end through this combined training objective.

E. Training

We trained the model for 1,000,000 steps with a batch size of 2048. We kept the learning rate constant at $5.4\text{e-}4$ for the first 350,000 steps, then linearly annealed it to $8\text{e-}6$. The forward kinematic loss weight λ_{FK} was weighted 6 to 1 to the flow matching weight λ_{FM} . For the FK loss, the position weight factor ω_p was weighted 5 to 1 relative to the orientation weight ω_o . We used the model’s exponential moving average (EMA) weights to reduce noise, as it is common to flow matching models. We evaluated the model with 10 flow-matching steps. For the experiments that only require a single solution, we inferred the model with a batch size of 16 and selected the best solution for further DLS optimization and reporting. All experiments were done on a single Nvidia RTX 5080 with 16GB of VRAM.

IV. EXPERIMENTS

In our experiments, we evaluate three different applications of the proposed model. We evaluate it as a raw IK solver, as a fine-tuned Foundation model for a single robot, as well as a seed model for other optimization approaches like DLS [19]. To show the model’s generalization capabilities, we evaluate it on two groups of robots: held-out robots, unseen during training but created by the same generator, and real-world robots, which were unseen during training. For the fine-tuning experiments, we randomly picked three real-world robots from each DoF and fine-tuned the model only on these samples. To show the model’s capabilities as a foundation model, we also trained the same model from scratch on these robots and report its performance after the same number of steps.

V. RESULTS

Table II shows the model’s performance on the different groups of evaluated robots. Since flow matching lets us sample multiple solutions for the same pose, we sample 16 configurations per pose and report the best, utilizing the flow matching model’s parallel inference and providing a realistic setup that prevents model failure. On average, synthetic robots reach a raw mean position error of 3.67% of reach and an orientation error of about 2.84° . The tail of the error distribution is about 2.5 times worse, with about 9.36% position and 7.58° orientation error. On the synthetic robots, 6- and 7-DoF robots are very close, with the higher-DoF robots performing slightly worse. On the real-world robots, all DoF perform similarly, with the 6-DoF robots performing slightly worse.

A single step of Damped Least Squares (DLS) optimization reduces the mean position error by a factor of 4 to below the 1% range and brings the mean orientation error to less than 1° . Optimizing the model output with DLS has a stronger effect for higher-DoF robots, reducing the mean and median error faster. Real-world 6-DoF robots keep a 0.6% reach error after one step, while the same robots with a wrist show about half that error. Polishing makes the error distribution more tail-heavy. 3 DLS steps are generally enough to bring the median error into the 0.01% range, while the 99th percentile stays high. This is especially true for the 6-DoF robots, which still have a 1.98% position error at the 99th percentile after 5 DLS steps.

Although the evaluated real-world robots are not sampled from the same distribution as the training data and are hand-designed by humans, the model’s solutions for these robots tend to be better than those for the random robots.

While the raw model output is comparable to the one for the random robots, the hand-crafted architectures show much larger gains from DLS optimization. A single step already brings the mean error of the real-world robots down to about half a centimeter, and 3 steps lead to sub-millimeter precision. This behavior is basically mirrored for the orientation.

Figure 5 shows the real robots’ performance on a per-robot level. For a fair comparison, we scaled the robots to unit

TABLE II: Performance of the different robot groups (best of 16): raw MorphIK result and after 1, 3, and 5 DLS steps

		raw		+1 DLS		+3 DLS		+5 DLS	
		pos	ori	pos	ori	pos	ori	pos	ori
position in % of reach, orientation in deg									
Heldout 6-DoF (200)	mean	3.434	2.72	0.737	0.41	0.221	0.09	0.145	0.08
	median	3.126	2.45	0.474	0.20	0.087	0.01	0.040	0.00
	p99	9.034	7.50	4.870	3.87	2.476	1.50	1.981	1.21
Heldout 7-DoF (200)	mean	3.446	2.65	0.625	0.41	0.149	0.08	0.089	0.07
	median	3.150	2.39	0.408	0.19	0.053	0.00	0.020	0.00
	p99	8.968	7.22	3.979	3.93	1.780	1.63	1.256	1.41
Heldout 8-DoF (200)	mean	3.704	2.79	0.579	0.42	0.101	0.06	0.051	0.04
	median	3.424	2.55	0.396	0.21	0.034	0.00	0.010	0.00
	p99	9.306	7.37	3.291	3.66	1.161	1.29	0.731	0.94
Heldout 9-DoF (200)	mean	4.102	3.21	0.606	0.49	0.074	0.04	0.030	0.03
	median	3.805	2.95	0.428	0.27	0.026	0.00	0.006	0.00
	p99	10.124	8.23	3.158	3.73	0.812	0.84	0.431	0.53
Heldout, all (800)	mean	3.671	2.84	0.637	0.43	0.136	0.07	0.079	0.05
	median	3.376	2.58	0.427	0.22	0.050	0.00	0.019	0.00
	p99	9.358	7.58	3.824	3.80	1.557	1.31	1.100	1.02
Real 6-DoF (14)	mean	4.216	3.34	0.599	0.39	0.078	0.02	0.041	0.01
	median	3.904	3.07	0.344	0.17	0.017	0.00	0.003	0.00
	p99	10.519	8.77	4.263	3.59	0.767	0.21	0.486	0.08
Real 7-DoF (8)	mean	3.999	3.40	0.323	0.26	0.016	0.00	0.007	0.00
	median	3.754	3.21	0.221	0.17	0.001	0.00	0.000	0.00
	p99	9.722	7.96	1.644	1.73	0.223	0.05	0.109	0.04
Real 8-DoF (14)	mean	3.936	2.95	0.297	0.24	0.013	0.01	0.005	0.01
	median	3.646	2.70	0.195	0.13	0.003	0.00	0.000	0.00
	p99	9.561	7.40	1.649	1.76	0.148	0.07	0.061	0.05
Real 9-DoF (8)	mean	3.965	3.11	0.205	0.17	0.005	0.00	0.001	0.00
	median	3.725	2.95	0.154	0.12	0.000	0.00	0.000	0.00
	p99	9.369	7.14	0.884	0.95	0.076	0.03	0.026	0.03
Real, all (44)	mean	4.042	3.18	0.381	0.28	0.033	0.01	0.016	0.01
	median	3.762	2.95	0.240	0.15	0.006	0.00	0.001	0.00
	p99	9.860	7.89	2.341	2.19	0.345	0.10	0.199	0.05
Heldout, all (random init) (800)	mean	42.087	57.02	29.044	43.47	12.266	19.95	5.027	7.72
	median	40.098	55.89	25.851	42.00	9.104	17.53	2.414	3.59
	p99	90.046	110.21	76.893	92.06	50.447	61.05	31.511	41.34
Real, all (random init) (44)	mean	53.105	58.93	35.966	44.89	13.754	21.09	4.476	7.80
	median	50.999	57.38	32.416	43.31	9.656	18.89	1.776	3.97
	p99	114.611	113.66	94.657	95.21	58.522	62.54	31.968	40.69

length, since robot chain length strongly affects the position error. Position-wise, the piper with the shadow hand performs best with 2.37%, and the Comaue e.DO with the shadow-hand wrist performs worst at 5.98% of reach. Orientation-wise, the best is again the piper at 1.69°, and the worst is ROBOTIS OMY-L100 at 4.78°. Overall, the arms perform very similarly with and without the attached wrist, as shown by the similarity between results 6- and 8-DoF and between 7- and 9-DoF. This suggests that most error comes from the arm, which makes sense since early joints in the chain have a stronger impact on overall position error than joints at the end of the chain. While the wrist does not seem to affect raw error much, the same is not true for the DLS-optimized outputs. When we compare the same arms with and without a wrist after 1 DLS step, the 8- and 9-DoF variants optimize faster than their 6- and 7-DoF counterparts.

A. Null space Modeling

Table III reports the spread of generated solutions for the same target pose. We sampled 256 random poses for each robot. For each pose, we generated 128 solutions and optimized them in 3 DLS steps. The spec shows the counts of solutions that land within 1% and 2°. We report variety measures only for in-spec solutions, since randomly failed solutions would be distinct and increase the standard deviation but carry no information about the model's null space exploration. $distinct_{0.25}$ is the number of solutions that differ by at least 0.25 radians in one joint compared to all other solutions; $std[rad]$ reports the standard deviation of the in-spec solutions.

6-DoF Manipulators are a special case in this evaluation, since they do not have a null space but a fixed set of up to 16 distinct redundant configurations for a given pose, as shown by [20]. For the synthetic robots, about half of the

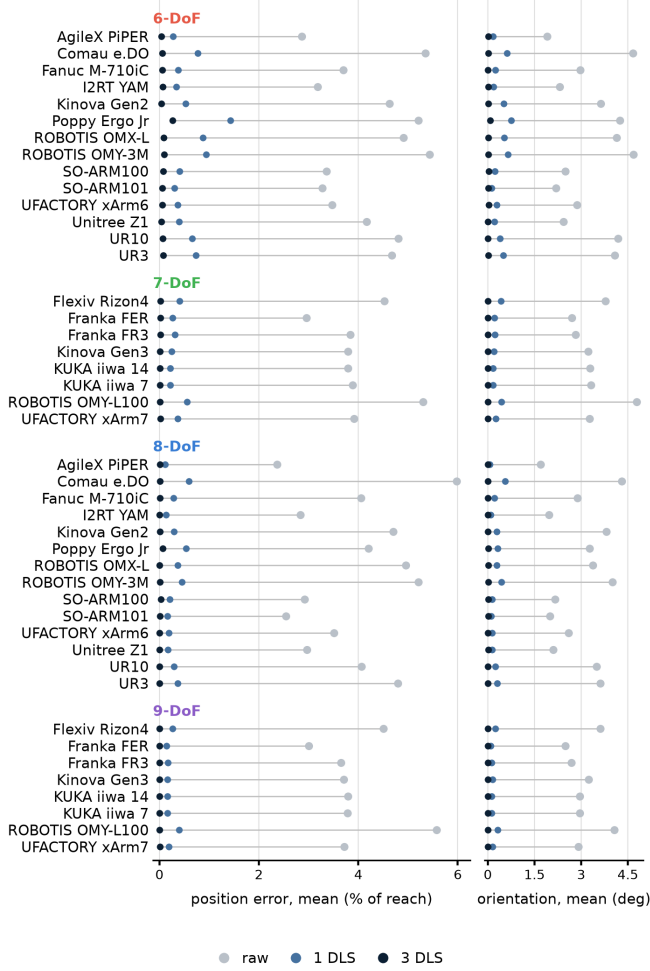


Fig. 5: Mean performance of unseen real-world robots on 512 random poses. Raw and after 1 and 3 steps.

samples per pose meet the 1 cm and 2° spec in 3 DLS steps. For 6-DoF, we found a mean of 3.4 different configurations per pose and a standard deviation of 0.28 rad. Absolute distinct counts as well as standard deviation increase with more degrees of freedom, which is expected and a general sign of a higher-dimensional null space of the higher-DoF chains.

The real robots show slightly different and better behavior. For the 6-DoF arms, the model finds 6.2 configurations on average. The number increases to 42.8 for the 7-DoF arms, which have a well-conditioned null space they can use. We also see an increase in in-spec solutions, which we suspect results from better manipulator conditioning that makes DLS more effective. The 8-DoF arms behave slightly worse than the set of 7-DoF arms. Since these were constructed by adding a wrist with smaller joint ranges to a 6-DoF arm, they are more limited in motion than the 7-DoF arms, although they have one more degree of freedom as shown in Table I. Adding a wrist to the 7-DoF arm greatly improves redundancy, yielding an average of 80.5 distinct configurations out of 96.1 in spec solutions. At this point,

TABLE III: Redundancy Spread for the same pose

	in spec / 128	distinct _{0.25}	std [rad]
<i>heldout synthetic robots, 128 samples per pose, 3 DLS steps</i>			
Heldout 6-DoF (200)	61.8	3.4	0.28
Heldout 7-DoF (200)	66.1	11.9	0.49
Heldout 8-DoF (200)	68.1	32.4	0.66
Heldout 9-DoF (200)	69.0	55.3	0.81
<i>real arms from, 128 samples per pose, 3 DLS steps</i>			
Real 6-DoF (14)	77.0	6.2	0.76
Real 7-DoF (8)	89.5	42.8	1.14
Real 8-DoF (14)	84.3	34.0	0.88
Real 9-DoF (8)	96.1	80.5	1.16

almost every solution is distinct, so the count is bounded by the number of samples instead of the model’s capabilities.

B. Finetuning

As shown above, our model generalizes well over a large variety of robot morphologies and provides a strong prior for post-inference optimization through DLS. As a second possible application, we propose using the model as an IK Foundation model that can be fine-tuned for a specific robot. While this approach reduces the model’s ability to generalize, we expect it to improve raw output performance on a given robot while keeping the training budget low compared to training from scratch. To test this capability, we randomly selected 8 real robots and trained the model only on these (one model per robot). In another validation setting, we also trained the model for these robots from scratch for the same number of steps.

Table IV shows the results of this experiment. Since these are aggregated numbers across multiple robots, we report the position error as a percentage of reach rather than in mm. The experiment shows that even a very small amount of fine-tuning can drastically reduce the error. After 50 steps, which corresponds to 1 second on the used hardware, the position error drops by 1.6% of reach and 1.1° , corresponding to about a 30% improvement in position and 25% in orientation. In that short time frame, the from-scratch model learns nothing. After 1000 steps, the position error is halved compared to the baseline model, while the from-scratch model still fails on orientation and slowly learns position. Generally, the gap between the pretrained and the from-scratch control shrinks with longer training. After 20,000, the gap reduced to a factor of 3, with the pretrained model reaching a mean positional error of 1.48% of reach and the control reaching

TABLE IV: Fine-tuning vs from scratch training

steps	time	pretrained		from scratch	
		pos [%]	ori [$^\circ$]	pos [%]	ori [$^\circ$]
0	—	5.58	4.65	52.49	57.62
50	1 s	3.91	3.55	52.14	57.77
300	4 s	3.11	3.09	39.72	53.66
1000	14 s	2.62	2.89	18.66	40.04
3000	42 s	2.17	2.53	11.80	30.18
20000	5 min	1.48	1.85	4.41	9.35

TABLE V: Inference Time for different batch sizes

poses	model [ms]	m. per pose [ms]	+3 DLS [ms]	select [ms]	total [ms]	per pose
1	3.5	3.5	5.0	1.3	9.8	9.85
8	4.1	0.5	5.1	1.3	10.4	1.30
64	13.5	0.2	5.1	1.3	19.9	0.31
512	82.7	0.16	5.1	1.3	89.1	0.17

4.41%. This is expected, since training gains slow over time and the control starts catching up to the fine-tuned model, although it does not reach it within the given time frame. This experiment shows that fine-tuning the model to a single robot can significantly improve performance, even with a tight budget. Compared to the from-scratch control group, it shows that the general model can serve as a strong starting point for training unseen robots. While it does not unlock capabilities that a from-scratch model could not learn, it gives any robot in this experiment a strong head start.

C. Time

Table V shows the model’s inference time per pose. DLS optimization and best-config selection add a constant time cost of 5 ms and 1.3 ms, respectively. DLS scales linearly with iterations at about 1.7 ms per step and is independent of batch size. Model inference is variable with the number of inferred poses. Inferring a single pose with a best-of-16 approach costs about 3.5 ms. This per-pose inference cost drops drastically when we infer multiple poses per batch: 8 poses cost 0.5 ms per pose, and batches with 512 elements cost about 0.16 ms per pose. The constant upfront cost of DLS and pose selection, plus relative speed gains from batched inference, make the approach ideal for sampling large batches of poses at once, e.g., for planning.

VI. CONCLUSION

MorphIK provides a new multi-robot architecture for robot-independent training. Trained only on procedurally generated robotic arms, MorphIK generalizes to a wide range of real 6- to 9-DoF robotic arms by learning to embed the morphological characteristics of kinematic chains. The proposed multi-robot architecture performs strongly across a wide range of robot morphologies, achieving a mean positional error of about 4% of the robots’ reach and a 3.2° orientation error without any further setup. Further, we showed that the generated joint configurations serve as excellent priors for further optimization approaches like DLS, reducing the residual error in 3 steps to sub-0.1% of reach and 1° range, which corresponds to a sub-1 mm position error on regular-sized arms, making the model viable and attractive even for high-precision tasks. One further main advantage of our model over traditional non-neural generative methods is its ability to explore the manipulators’ null space in parallel. In future work, we will examine the learned embedding space for different tasks and explore a transformer-based decoder to make the model truly DoF-independent.

VII. ACKNOWLEDGMENT

The authors produced and validated all scientific content. AI-based tools, including Grammarly and Claude Opus 5 (Anthropic), were used to develop auxiliary code and check grammar.

REFERENCES

- [1] B. Ames, J. Morgan, and G. Konidakis, “IKFlow: Generating diverse inverse kinematics solutions,” *CoRR*, vol. abs/2111.08933, 2021.
- [2] J.-G. Habekost, C. Gäde, P. Allgeuer, and S. Wermter, “Inverse kinematics for neuro-robotic grasping with humanoid embodied agents,” in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 7315–7322.
- [3] Z. Zhang and Z. Jiao, “IKDiffuser: a diffusion-based generative inverse kinematics solver for kinematic trees,” 2026.
- [4] J. Yang, S. Yan, F. Feng, C. Yao, G. Wang, Z. Mai, Y. Zhao, and Y. Han, “MimicIK: Real-time generative inverse kinematics from teleoperation with fk consistency,” *arXiv preprint arXiv:2606.15148*, 2026.
- [5] R. Smits, “KDL: Kinematics and Dynamics Library,” <http://www.orocos.org/kdl>.
- [6] P. Beeson and B. Ames, “TRAC-IK: An open-source library for improved solving of generic inverse kinematics,” in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, 2015, pp. 928–935.
- [7] S. Starke, “Bio IK: A memetic evolutionary algorithm for generic multi-objective inverse kinematics,” Ph.D. dissertation, Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky, 2020.
- [8] O. Limoyo, F. Marić, M. Giamou, P. Alexson, I. Petrović, and J. Kelly, “Generative graphical inverse kinematics,” *IEEE Transactions on Robotics*, vol. 41, p. 1002–1018, 2025.
- [9] T. Yoshikawa, “Manipulability of robotic mechanisms,” *The International Journal of Robotics Research*, vol. 4, no. 2, pp. 3–9, 1985.
- [10] J. Angeles and C. S. López-Cajún, “Kinematic isotropy and the conditioning index of serial robotic manipulators,” *The International Journal of Robotics Research*, vol. 11, no. 6, pp. 560–571, 1992.
- [11] R. W. Brockett, “Robotic manipulators and the product of exponentials formula,” in *Mathematical Theory of Networks and Systems: Proceedings of the MTNS-83 International Symposium Beer Sheva, Israel, June 20–24, 1983*. Springer, 2005, pp. 120–129.
- [12] S. Caron, G. Romualdi, L. Kozlov, D. F. Ordoñez Apraez, H. Tadashi Kussaba, S. H. Bang, K. Zakka, F. Schramm, J. Uruç, S. Traversaro, J. Zamora, S. Castro, H. X. Tao, J. Yu, W. Jallet, Y. Zhang, N. Walker, N. Bragin, and W. Wie, “robot_descriptions.py: Robot descriptions in Python,” 2026.
- [13] D. L. Pieper, *The kinematics of manipulators under computer control*. Stanford University, 1969.
- [14] A. Gupta, L. Fan, S. Ganguli, and L. Fei-Fei, “Metamorph: Learning universal controllers with transformers,” in *International Conference on Learning Representations*, 2022.
- [15] Y. Lipman, R. T. Q. Chen, H. Ben-Hamu, M. Nickel, and M. Le, “Flow matching for generative modeling,” in *The Eleventh International Conference on Learning Representations*, 2023.
- [16] S. Peebles, William & Xie, “Scalable diffusion models with transformers,” in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, 2023, pp. 4172–4182.
- [17] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter *et al.*, “ π_0 : A vision-language-action flow model for general robot control,” *arXiv preprint arXiv:2410.24164*, 2024.
- [18] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.
- [19] C. W. Wampler, “Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 16, no. 1, pp. 93–101, 1986.
- [20] M. Raghavan and B. Roth, “Inverse kinematics of the general 6r manipulator and related linkages,” 1993.