Real-Time Robot Self-Modeling via Direct Sensorimotor Decoding: Overcoming the Neural Rendering Latency Bottleneck

Muhammad Zeeshan Asghar
Master's Program in Data Science
Higher School of Economics
Moscow, Russian Federation
Email: masgar@edu.hse.ru

***Abstract*—Autonomous robot self-modeling—the ability of a robot to learn its own morphology from sensory data—is crucial for resilience in unstructured environments. While recent advances in neural rendering, particularly Neural Radiance Fields (NeRFs), have enabled high-fidelity 3D self-models, their computational latency (typically $<$30 FPS) precludes real-time deployment in control loops. This paper presents a comprehensive comparative study of four self-modeling architectures, introducing two novel approaches that overcome this latency bottleneck.**

Implementation Scope: We reproduce from scratch the Free-Form Kinematic Self-Model (FFKSM), a NeRF-based architecture achieving 17.35 dB PSNR at 5.22 FPS on our custom dataset. We then develop Kinematic 3D Gaussian Splatting (K-3DGS), an explicit representation that improves speed to 37 FPS but suffers from "blobby" artifacts (17.01 dB PSNR) due to isotropic Gaussian primitives. Our primary contribution is NeuroKin—a lightweight, fully convolutional network that directly decodes joint angles to visual silhouettes, achieving superior quality (21.88 dB PSNR) at over $200\times$ higher speed (7400 FPS, 0.135 ms latency). We further introduce ResNeuroKin-D, a multi-task variant that learns geometrically structured latent representations through synthetic depth prediction, maintaining high quality (21.23 dB PSNR) at 2500 FPS. These direct sensorimotor decoders enable novel real-time applications including proprioceptive drift correction and rapid damage adaptation. All code is available at https://github.com/YoloPopo/robot_self_modelling.

***Index Terms*—Robot self-modeling, neural rendering, real-time inference, 3D Gaussian splatting, sensorimotor decoding, damage recovery, proprioceptive drift**

I. Introduction

Autonomous robots operating in remote environments face a fundamental paradox: they must adapt to physical damage without human intervention, yet traditional control systems assume a static, predefined morphology. When a Mars rover's linkage bends or a disaster-response robot loses a joint, the divergence between its internal kinematic model and physical reality leads to catastrophic control failure. Recent advances in neural self-modeling have enabled robots to learn their own morphology from visual feedback, but these approaches suffer from two critical bottlenecks: computational latency that precludes real-time control, and catastrophic forgetting when adapting to damage. This work demonstrates that by rethinking the fundamental approach to self-modeling—bypassing explicit 3D reconstruction in favor of direct sensorimotor decoding—we achieve a $1418\times$ speedup with superior visual fidelity, enabling sub-millisecond latency suitable for high-frequency control loops.

A. Problem Statement

Robots deployed in remote, unstructured environments—planetary rovers, disaster response robots, deep-sea manipulators—must operate autonomously for extended periods without human intervention. A critical capability for such autonomy is *self-modeling*: the ability to learn and maintain an accurate internal representation of one's own morphology. Traditional robotic systems rely on predefined kinematic models (URDF files) that become invalid when the robot suffers damage, wear, or mechanical deformation. This "reality gap" between the internal model and physical state leads to catastrophic control failures.

Recent advances in neural rendering have enabled data-driven self-modeling approaches. While achieving impressive visual fidelity, NeRF-based methods share a fundamental limitation: they require hundreds of network evaluations per pixel during volumetric rendering, resulting in prohibitive computational latency. On a modern GPU (NVIDIA T4), the Free-Form Kinematic Self-Model (FFKSM) achieves only 5.22 Frames Per Second (FPS), far below the required FPS for integration into high-frequency control loops (1 kHz) common in robotic systems.

B. Motivation and Course Context

This work implements and extends the methodology from "Teaching Robots to Build Simulations of Themselves" by Hu et al. [1], presented at Nature Machine Intelligence 2024. The original paper introduces Free-Form Kinematic Self-Models

(FFKSM), adapting Neural Radiance Fields [2] for robot self-modeling. We provide a complete independent implementation, revealing practical challenges and proposing efficiency improvements.

C. Research Questions

This project investigates four research questions that structure our experimental methodology. First, *reproducibility*: can FFKSM be successfully reproduced, and what implementation challenges arise in practice? Second, *architectural trade-offs*: how do explicit 3D Gaussian Splatting versus implicit NeRF versus direct regression approaches compare on speed-accuracy frontiers? Third, *data efficiency*: how does limited dataset scale (2,000 versus 12,000 samples) affect different architectural paradigms? Fourth, *real-time feasibility*: can self-modeling achieve 1 kHz control-loop compatibility ($\geq$1000 FPS) necessary for robotic deployment?

D. Contributions

This project makes five primary contributions to the field of robotic self-modeling:

1) **Complete independent implementation of FFKSM** from scratch, documenting critical implementation challenges including black-screen convergence and kinematic blindness bugs.
2) **Development of Kinematic 3D Gaussian Splatting (K-3DGS)**, an explicit representation using 3D Gaussians attached to a differentiable kinematic chain, achieving 37 FPS but revealing limitations of isotropic primitives for thin articulated structures (17.01 dB PSNR).
3) **NeuroKin**, our primary contribution—a lightweight convolutional network that directly maps joint angles to visual silhouettes, achieving superior quality (21.88 dB PSNR) at over 200$\times$ higher speed (7400 FPS versus FFKSM's 5.22 FPS), demonstrating that 3D reconstruction is unnecessary for many self-modeling tasks.
4) **ResNeuroKin-D**, a multi-task learning extension that learns structured latent representations through joint silhouette and synthetic depth prediction, maintaining high quality (21.23 dB PSNR) at 2500 FPS.
5) **Reproducible dataset generation pipeline** using Lorenz attractor trajectories for smooth workspace exploration, generating 2,000 diverse robot configurations with documented statistics and hyperparameters.

E. Scope Clarification

To set appropriate expectations for course evaluation, we explicitly delineate the scope of this work. This project **is**:

- An independent implementation of FFKSM
- A comparative study of four architectural paradigms
- A demonstration of over 200$\times$ speedup via direct decoding
- An original contribution of three novel architectures (K-3DGS, NeuroKin, ResNeuroKin-D)

This project **is not**:

- A direct reproduction of original FFKSM experiments (we use a different dataset: 2,000 versus 12,000 samples)
- Real robot validation (evaluation remains simulation-based in PyBullet)
- A full anisotropic 3DGS implementation (we simplified to isotropic Gaussians due to optimization challenges)
- A deployment study with actual control integration

Our work demonstrates that for many practical robotic tasks requiring only silhouette-level self-awareness (collision checking, drift correction), explicit 3D reconstruction via neural rendering is computationally excessive. Direct sensorimotor decoders provide comparable—or superior—visual fidelity at over 200$\times$ higher speed, making real-time self-modeling feasible on embedded hardware.

II. Related Work

The capacity for autonomous adaptation to physical damage is the defining characteristic of biological resilience, yet it remains an elusive goal for robotic systems. This section traces the evolution of robotic self-modeling through three distinct eras—symbolic, behavioral, and visual—before examining the computational bottlenecks and theoretical challenges that motivate our direct decoding approach.

A. The Evolution of Robotic Self-Modeling

1) The Symbolic Era: Estimation-Exploration: The foundational work in robotic self-modeling is Bongard et al.'s seminal paper ”Resilient Machines Through Continuous Self-Modeling” [3], which introduced a co-evolutionary algorithm wherein a four-legged modular robot simultaneously evolved populations of kinematic hypotheses (”models”) and diagnostic motor commands (”tests”). When the robot suffered damage—such as removal of a leg segment—it lacked direct sensors to detect the loss. Instead, it executed test actions, compared real sensor data (tilt and IMU readings) against model predictions, and pruned hypotheses that failed to explain observed behavior. The core innovation was an active learning loop: the fitness of a test action was determined by its ability to cause disagreement among the best candidate models, thereby maximally reducing uncertainty. After approximately 16 modeling-testing cycles spanning several minutes of real-world experimentation, the robot converged on an accurate damaged model and synthesized a compensatory gait through reinforcement learning on the internal simulation.

While conceptually elegant, this approach suffers from the *symbolic bottleneck*: the robot can only discover models constructible from predefined physics engine primitives (cylinders, hinge joints). It cannot represent non-rigid deformations, soft-body dynamics, or arbitrary visual damage—limitations that become critical in unstructured environments where failure modes are unpredictable. As noted by Kwiatkowski and Lipson [4], the search space of symbolic physics engines is discrete and non-differentiable, making optimization slow and prone to local optima. Furthermore, the approach relied heavily on simple proprioceptive sensors providing limited information compared to high-bandwidth vision.

2) The Behavioral Era: Avoidance Over Adaptation: Recognizing the difficulty of explicit physical modeling, Cully et al. [6] introduced Intelligent Trial and Error (IT&E), which bypassed damage diagnosis entirely. Their key insight was to focus on *finding what still works* rather than diagnosing what went wrong. Using the Map-Elites algorithm [7], they pre-computed a "behavioral repertoire" of approximately 13,000 distinct gaits (requiring 40 million simulations over two weeks of computation). This repertoire is a high-dimensional grid where each cell corresponds to a specific behavioral descriptor (e.g., duty cycle versus leg offset) and stores the highest-performing controller found for that descriptor. When damaged in the field, the robot treated this map as a Bayesian prior, using Gaussian Process regression to search for a behavior that performs well in the real world. By testing a few dozen distinct behaviors, the Gaussian Process updated its posterior mean and variance, rapidly identifying a compensatory gait. Cully et al. demonstrated recovery in less than two minutes of real-world testing across various damage scenarios, including removal of entire legs.

However, IT&E represents a strategy of *avoidance*, not adaptation. The robot does not repair its internal model—it discards the parts of the behavioral space that no longer function. While effective for locomotion, this limits utility in manipulation tasks requiring precise kinematic accuracy. If a robot needs to reach through a narrow aperture, it cannot simply "try a different gait"—it must understand the precise geometry of its damaged arm. Furthermore, as robotic complexity increases, the curse of dimensionality makes pre-computing dense behavioral maps prohibitively expensive: a 6-DOF arm with even modest discretization could require billions of map entries.

3) The Visual Era: Deep Self-Models: The advent of convolutional neural networks enabled robots to model themselves directly in pixel space, bypassing the limitations of symbolic physics engines. Kwiatkowski and Lipson [4] proposed the concept of *task-agnostic self-modeling*, training deep forward models to predict the future state of the robot given the current state and action. This represented a pivotal shift from symbolic physics to learnable physics.

This trajectory culminated in the work of Chen et al. [5] and arguably the current state-of-the-art, Hu et al. [1]. Hu et al. proposed the Free-Form Kinematic Self-Model (FFKSM). Unlike previous methods that predicted future states, FFKSM learns to *query morphology*: given joint angles $\boldsymbol{\theta}$ and a 3D query point $\mathbf{x}$, the network outputs occupancy probability:

$$P(\text{occupied}|\mathbf{x}, \boldsymbol{\theta}) = f(\mathbf{x}, \boldsymbol{\theta}) \tag{1}$$

This allows the robot to generate "mental images" of itself from any viewpoint. The training is self-supervised: the robot observes itself via a fixed camera, compares rendered predictions to real observations, and backpropagates the error. Hu et al. reported that FFKSM achieves a model size of only 333 KB while maintaining high visual fidelity, representing a significant reduction from the 1.1 MB required by Chen et al.'s multi-camera approach [5].

While FFKSM removes dependence on URDF files and symbolic priors, it relies on a dense multi-layer perceptron that introduces two critical flaws. First, *global density*: the model is monolithic, meaning that "healing" the model (updating parameters) requires gradient updates that affect all parameters. This lacks the biological modularity required for local repair—if the robot bends its left arm, updates to the network often degrade the representation of the right arm. Second, *computational latency*: volumetric rendering requires hundreds of network evaluations per pixel, as we detail in the next subsection. Recent work by Hu et al. [12] demonstrated that applying articulated 3D Gaussian Splatting can improve visual fidelity of these models, but their approach still relies on a dense kinematic network to predict Gaussian parameters, leaving the catastrophic interference problem unsolved.

B. Neural Rendering: The Speed-Quality Tradeoff

The computational bottleneck of visual self-modeling stems fundamentally from the choice of 3D representation. Neural Radiance Fields (NeRFs) [2] revolutionized computer vision by representing scenes as continuous functions $F(\mathbf{x}, \mathbf{d}) \to (\mathbf{c}, \sigma)$, mapping 5D coordinates (3D spatial position $\mathbf{x}$ and 2D viewing direction $\mathbf{d}$) to color $\mathbf{c}$ and volume density σ. To render a single pixel, NeRF approximates the volume rendering integral via quadrature along a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, where $\mathbf{o}$ is the camera origin and $\mathbf{d}$ is the ray direction. The color $C(\mathbf{r})$ is computed as:

$$C(\mathbf{r}) = \sum_{i=1}^{N} T_i (1 - \exp(-\sigma_i \delta_i)) \mathbf{c}_i \tag{2}$$

where $T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right)$ represents accumulated transmittance (the probability that the ray has not hit a particle before sample i), and δ_i is the distance between adjacent samples. For a standard 100×100 image with $N = 64$ samples per ray, this formulation requires $100 \times 100 \times 64 = 640,000$ network evaluations per frame. At 5.22 FPS, this amounts to 3.2 million MLP evaluations per second—a computational burden that our subsequent implementations directly address.

To capture high-frequency geometric details, NeRF employs *positional encoding*, mapping input coordinates into a higher-dimensional space using sinusoidal functions:

$$\gamma(\mathbf{p}) = \left[\sin(2^0 \mathbf{p}), \cos(2^0 \mathbf{p}), \ldots, \sin(2^{L-1} \mathbf{p}), \cos(2^{L-1} \mathbf{p})\right] \tag{3}$$

While effective for static scenes, this encoding introduces significant computational overhead. Even with acceleration structures like Instant-NGP's hash encoding [23] or Mip-NeRF's scale-aware anti-aliasing [24], the latency remains too high for tight control loops (∼5.22 Hz) required in robotic damage recovery. Adapting NeRFs to articulated bodies via D-NeRF approaches [25] adds time-deformation networks, further compounding computational cost and implementation complexity.

Kerbl et al. [8] introduced 3D Gaussian Splatting (3DGS) as an explicit alternative to implicit NeRFs, representing scenes as discrete 3D Gaussians with learnable positions $\boldsymbol{\mu}_i \in \mathbb{R}^3$, covariances Σ_i (parameterized by scaling $\mathbf{S}$ and rotation $\mathbf{R}$), opacities $\alpha_i \in [0,1]$, and view-dependent colors encoded via spherical harmonics. The key innovation is the EWA (Elliptical Weighted Average) splatting algorithm, which projects 3D Gaussians into 2D screen space. The 3D covariance is projected to 2D covariance using the affine viewing transformation $\mathbf{W}$ and the Jacobian of the projection $\mathbf{J}$:

$$\Sigma' = \mathbf{J}\mathbf{W}\Sigma\mathbf{W}^T\mathbf{J}^T \tag{4}$$

This formulation enables rasterization-based rendering rather than ray-marching, avoiding the expensive sampling of empty space inherent in NeRFs. The GPU sorts Gaussians by depth using radix sort and alpha-blends them in a single pass. Kerbl et al. demonstrated rendering speeds exceeding 100 frames per second at 1920×1080 resolution on desktop GPUs (NVIDIA RTX 3090) for synthetic scenes. Matsuki et al. [9] and Keetha et al. [10] extended 3DGS to SLAM, proving robustness for real-time geometry reconstruction in dynamic environments.

However, embedded deployment remains challenging. Lassanske et al. [11] conducted detailed benchmarks of 3DGS on mobile hardware, finding that the parallel radix sort required for alpha-blending is memory-bandwidth bound on devices like NVIDIA Jetson Orin. The Jetson Orin operates under stricter power (15W typical) and memory bandwidth constraints (204 GB/s) compared to the RTX 3090's 936 GB/s. For self-repairing robots, the rendering pipeline must support not just inference but optimization (training on edge hardware). A standard 3DGS scene may contain millions of Gaussians—to run evolutionary algorithms for damage recovery, the robot must render hundreds of candidate morphologies per second. This necessitates a move beyond dense Gaussian fields toward sparse, budgeted representations.

C. The Plasticity-Stability Dilemma

The core challenge of damage recovery in neural systems is the *plasticity-stability dilemma*, first articulated by Grossberg [13]: the system must be plastic enough to learn new configurations (damage) yet stable enough to retain knowledge of undamaged components. In biological brains, this balance is achieved through *synaptic pruning*—the physical restructuring of the connectome wherein weak neuronal connections are eliminated while frequently used connections are strengthened. Hebbian learning ("neurons that fire together, wire together") ensures that functional modules (e.g., motor cortex regions for hand versus foot) are physically distinct. If a biological organism suffers a lesion in the foot region, the sparsity of the brain ensures the hand region remains largely unaffected. This structural modularity is what artificial neural networks typically lack.

Research by Chechik et al. [14] demonstrated through computational models that synaptic pruning is an optimal strategy *strictly under metabolic or resource constraints*. Importantly, they proved that in the absence of such energetic costs, pruning cannot improve network performance compared to an intact network—a finding that aligns precisely with edge compute constraints in robotics, where power budgets (15W typical for Jetson Orin) and memory bandwidth are severely limited. The biological precedent suggests that sparse, modular architectures may be not merely beneficial but necessary for resource-constrained adaptive systems.

Artificial neural networks, by contrast, typically employ fully connected architectures where catastrophic interference is well documented. McCloskey and Cohen [15] demonstrated that backpropagation in distributed (dense) representations leads to global weight changes. When a network trained on Task A is subsequently trained on Task B, weights shift to a new manifold, often destroying performance on Task A. In the context of robotic self-modeling, consider a robot with a damaged leg. To update its visual model, it collects new data pairs $(\boldsymbol{\theta}_{\text{leg}}, I_{\text{damage}})$ and minimizes prediction error via gradient descent:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} - \eta \nabla \mathcal{L}(I_{\text{damage}}, f(\boldsymbol{\theta}_{\text{leg}})) \tag{5}$$

Because hidden layers are shared across all joints, this gradient update changes weights encoding the kinematics of healthy arms. The robot learns its leg is broken but simultaneously forgets how to reach with its arm—an unacceptable failure mode for resilient systems that must maintain operational capabilities in undamaged subsystems.

Kirkpatrick et al. [16] proposed Elastic Weight Consolidation (EWC) to mitigate catastrophic forgetting. EWC takes a Bayesian perspective, approximating the posterior distribution of weights given previous task data using the Laplace approximation. The method augments the loss function for the new task with a quadratic penalty that anchors weights to their previous values, scaled by importance:

$$\mathcal{L} = \mathcal{L}_B + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta_{A,i})^2 \tag{6}$$

where $\theta_{A,i}$ are the optimal parameters for the previous task, and F_i is the diagonal of the Fisher Information Matrix (FIM). The Fisher Information estimates the curvature of the loss landscape: a high value indicates that the loss increases sharply if this parameter is changed, signaling importance.

While EWC is effective for sequential classification tasks (e.g., split MNIST), it is ill-suited for robotic damage recovery for two reasons. First, *computational cost*: calculating the Fisher Matrix requires computing second-order derivatives (Hessian) or approximating them via squared gradients. For high-dimensional rendering networks (millions of parameters), storing and computing $\mathbf{F}$ is prohibitively expensive for real-time adaptation on edge hardware. Second, *lack of modularity*: EWC slows forgetting but does not enable the robot to isolate the damage. It treats the network as a monolith with varying

”stiffness”. If damage is severe, the robot needs to change high-importance weights, but EWC prevents this, leading to under-fitting of the damage (the ”plasticity gap”). Research by Mermillod et al. [17] highlights parameter-based modulations, but these solutions remain within the paradigm of parameter tuning rather than structural reconfiguration. For the specific demands of robotic damage recovery—where physical modularity must be preserved under real-time constraints—such parameter-centric approaches may prove insufficient.

The failure of weight-based regularization suggests that *topology*, rather than weights, may hold the key to adaptive resilience. Frankle and Carbin [18] formalized the value of sparsity in deep learning through the *Lottery Ticket Hypothesis*: ”A randomly-initialized dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.” This suggests that the vast majority of weights in a dense self-model are redundant. Han et al. [19] empirically demonstrated this, achieving compression ratios of 49× on AlexNet and 39× on VGG-16 through a combination of pruning, quantization, and Huffman coding. Model storage reduced from 240 MB to 6.9 MB while maintaining baseline accuracy. Mocanu et al. [20] extended this with Sparse Evolutionary Training (SET), which maintains sparse topology throughout training rather than training dense and pruning post-hoc. In each epoch, SET prunes the weakest connections and regrows random new ones, allowing the network to explore the topology space dynamically.

For robotics, sparsity offers dual benefits. First, *efficiency*: sparse matrix operations reduce computational latency, directly addressing the rendering bottleneck. Second, *modularity*: by enforcing sparsity, the network is forced to develop modular sub-structures. Damage to one module can be repaired by updating only the weights within that module, significantly reducing catastrophic interference. This biological inspiration—sparse, modular connectivity enabling localized plasticity—motivates our architectural explorations in subsequent sections.

D. The Gap: Articulated Self-Modeling at Control-Loop Rates

While 3D Gaussian Splatting solves the speed bottleneck for static scene reconstruction, and evolutionary architecture search methods (e.g., NEAT [21], SPOS [22]) enable topology optimization for classification tasks, no prior work addresses articulated self-modeling with sub-millisecond latency suitable for high-frequency (1 kHz) control loops. FFKSM achieves high visual fidelity but remains far below 30 FPS. 3DGS-based SLAM systems achieve 100+ FPS but assume static geometry. Recent articulated 3DGS work [12] improves quality but retains the dense network bottleneck, achieving only modest speedups.

This gap motivates our central research question: can *direct sensorimotor decoding*—bypassing explicit 3D reconstruction entirely—achieve both superior speed and quality by aligning architectural inductive biases with the task structure? Our work demonstrates that for many practical robotic tasks requiring only silhouette-level self-awareness (collision checking, drift correction, damage detection), the answer is affirmative. The following sections detail our implementation journey, which progresses from faithful reproduction of FFKSM’s volumetric approach, through explicit 3DGS-based alternatives, to ultimately arrive at direct decoders that achieve over $200\times$ speedup with improved visual fidelity.

III. Understanding the Original FFKSM Paper

Before detailing our implementations, we provide a technical analysis of the FFKSM architecture to demonstrate understanding of the original contribution and motivate our subsequent design choices.

A. Core Innovation of FFKSM

The FFKSM paper [1] addresses a fundamental challenge in robot self-modeling: learning morphology without predefined CAD models. Previous approaches [4] required multiple calibrated depth cameras, introducing hardware complexity and calibration brittleness. FFKSM’s key insight is adapting NeRF’s implicit volumetric representation to articulated robots by conditioning occupancy on joint angles. This formulation enables the robot to ”imagine” its body from any viewpoint given only proprioceptive feedback (joint encoder readings).

The *virtual frame prior* introduced by Hu et al. transforms 3D query points using the first two joints (θ_0, θ_1) before network processing:

$$\mathbf{x}_{\text{virtual}} = R_z(-\theta_0)R_y(-\theta_1)\mathbf{x}_{\text{world}} \tag{7}$$

where R_z and R_y are rotation matrices about the z and y axes respectively. This transformation reduces the network’s learning burden by aligning coordinates with the robot’s base orientation, effectively factoring out base rotation from the end-effector position learning problem. Without this prior, the network would need to learn the circular symmetry of rotations—a challenging task for MLPs despite positional encoding.

The *split encoder architecture* separates coordinate processing (”where in 3D space?”) from kinematic processing (”which joint configuration?”). Two parallel encoders process: (1) a *coordinate encoder* that takes virtual-frame 3D points via positional encoding (5 frequencies $\rightarrow$ 33 dimensions) followed by an MLP, and (2) a *kinematic encoder* that processes remaining joints (θ_2, θ_3) via separate positional encoding (2×2 dimensions) and MLP. Fusion happens in a ”predictive module” outputting density σ and visibility v. This architectural split is critical: a single encoder processing $(\mathbf{x}, \boldsymbol{\theta})$ jointly would need to relearn the kinematic transformation for every 3D point. Splitting allows the kinematic encoder to learn once the mapping joint angles $\rightarrow$ link transformations, then apply it to all query points, dramatically improving sample efficiency.

Positional encoding maps coordinates into higher-frequency space via $\gamma(\mathbf{p}) = [\sin(2^0\mathbf{p}), \cos(2^0\mathbf{p}), \ldots, \sin(2^{L-1}\mathbf{p}), \cos(2^{L-1}\mathbf{p})]$ with $L = 5$ frequencies. Without high-frequency basis functions, MLPs struggle to represent fine geometric details such as sharp link edges and joint boundaries, due to the spectral bias of neural networks toward low frequencies. The positional encoding provides sufficient frequency content for capturing geometric detail while remaining computationally tractable.

B. Computational Analysis and Limitations

For each camera ray, FFKSM samples $M = 64$ points and computes pixel intensity via volume rendering:

$$I_{ij} = \sum_{m=1}^{M} \alpha_m v_m \sigma_m \times \left(1 - \exp\left(-\mathrm{ReLU}(\sigma_m) \sum_{n=1}^{m-1} \exp(-\mathrm{ReLU}(\sigma_n))\right)\right) \quad (8)$$

where α_m is the accumulated alpha value accounting for transmittance. This requires 640,000 network evaluations per 100×100 image (10,000 rays $\times$ 64 samples), explaining the fundamental computational bottleneck. At 5.22 FPS, this amounts to 3.3 million evaluations per second. Even with efficient batching and GPU parallelization, the sheer number of serial network evaluations along each ray creates an irreducible latency floor.

The original paper identified several limitations relevant to our work. First, *computational cost*: ∼5.22 FPS on Tesla T4 limits real-time use in robotic control loops operating at 100+ Hz. Second, *dataset scale*: FFKSM required 12,000 samples for convergence, representing several hours of data collection—potentially dangerous for damaged robots. Third, *single viewpoint*: training on a fixed camera limits generalization to novel viewpoints, restricting deployment flexibility. Fourth, *simulation-only validation*: all experiments occurred in PyBullet without real robot deployment, leaving the sim-to-real gap unaddressed.

Our implementation directly confronts these limitations while uncovering additional challenges (black-screen convergence, class imbalance, kinematic blindness), as detailed in the methodology section.

IV. Methodology

Our implementation journey comprised four distinct phases: dataset generation using chaotic attractors for smooth workspace exploration, independent reproduction of FFKSM revealing undocumented implementation challenges, development of an explicit 3DGS-based alternative exposing the expressiveness-stability tradeoff, and ultimately the realization that direct sensorimotor decoding could bypass 3D reconstruction entirely. This section narrates each phase as a progression of insights that culminated in our primary contribution.

A. Dataset Generation via Lorenz Attractors

The original FFKSM paper generated training data using smooth trajectories. Rather than random "motor babbling"—which produces jerky, discontinuous motions that poorly sample the workspace and introduce unnatural artifacts during training—we employed trajectories generated by the Lorenz dynamical system. This chaotic attractor, defined by the coupled differential equations:

$$\begin{aligned} \frac{dx}{dt} &= \sigma(y - x), \\ \frac{dy}{dt} &= x(\rho - z) - y, \\ \frac{dz}{dt} &= xy - \beta z \end{aligned} \quad (9)$$

with parameters $\sigma = 10$, $\rho = 28$, $\beta = 8/3$, produces deterministic yet non-repeating trajectories that ergodically explore the state space. We integrated these equations using fourth-order Runge-Kutta with time step $\Delta t = 0.01$, scaling outputs to the robot's joint limits (±90°). Two independent attractors drove the four joints, ensuring diverse poses without unnatural velocity discontinuities that would corrupt the learned model's smoothness assumptions.

The resulting dataset comprises 2,000 samples (1,600 training, 400 test), each containing: a 100×100 grayscale silhouette (binary mask), four joint angles (in degrees), and fixed camera parameters (position [1.0, 0.0, 0.0], focal length 130.2545). This scale—one-sixth that of the original paper's 12,000 samples—reflects computational constraints but enables fair comparison across methods while revealing how architectural choices impact data efficiency. Statistical analysis confirms good workspace coverage: joint angle means near zero (-0.23° to 12.45°), standard deviations around 30°, and robot occupancy averaging 14.7% of pixels (range 8.9%–23.4%), creating the severe class imbalance that later motivated our curriculum learning solution.

B. Free-Form Kinematic Self-Model: Implementing the Baseline

We implemented FFKSM from scratch. The architecture employs a split encoder design (as described previously): 3D query points undergo virtual frame transformation using the first two joint angles, then separate coordinate and kinematic encoders process spatial and joint information respectively. The predictive module concatenates features and outputs density σ and visibility v for volumetric rendering.

1) Challenge 1: Black-Screen Convergence: Initial training exhibited a pathological failure mode: the network converged to predicting uniformly black images (all zeros) despite achieving low loss values. Analysis revealed the root cause—robots occupy only $\approx 15\%$ of image pixels, creating severe class imbalance (85% background). The network discovered a local minimum where predicting the majority class (black) achieved mean squared error of ∼0.85, yet produced no useful representation. This failure persisted for the first 1,000 iterations, rendering standard training completely ineffective.

Standard remedies proved inadequate. Weighted loss functions (applying 5× penalty to robot pixels) provided marginal improvement but introduced training instability and slow convergence. Focal loss, designed to address class imbalance in object detection, actually worsened the problem by introducing

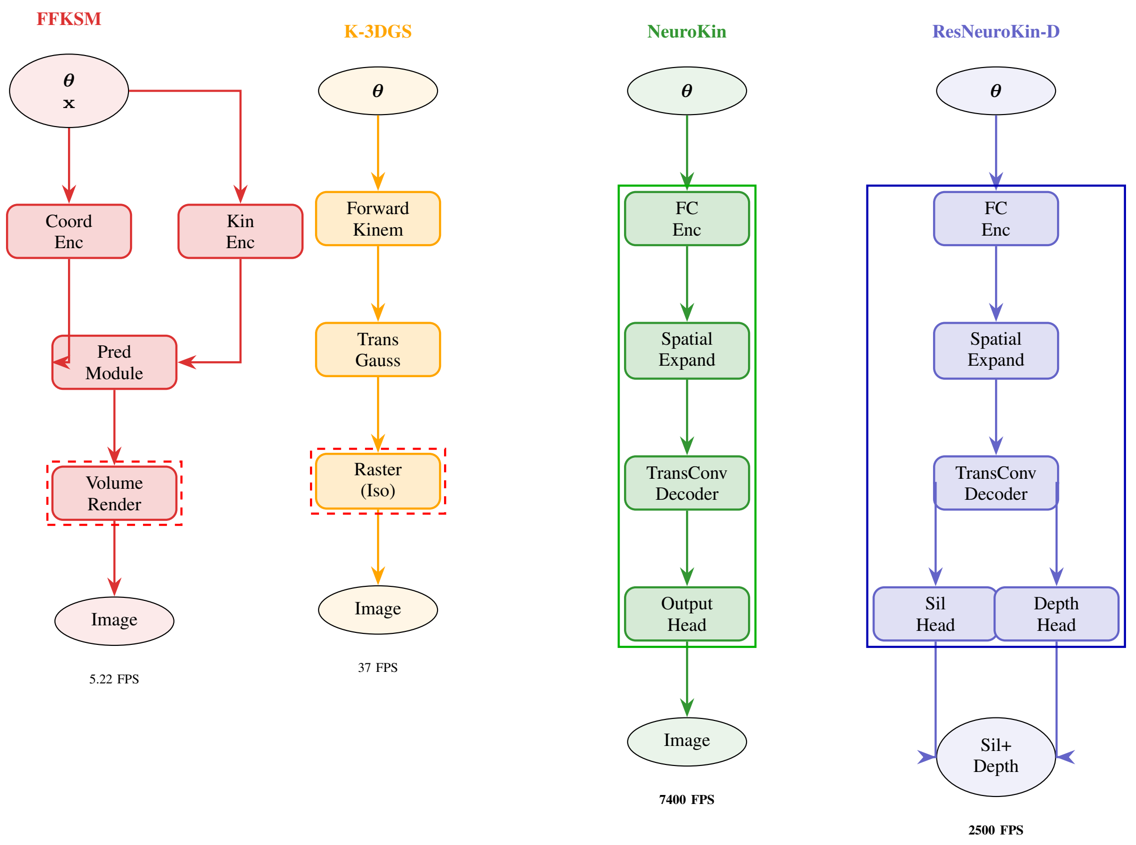

Fig. 1. Architectural comparison of self-modeling approaches. **FFKSM (leftmost):** Volumetric rendering requiring 640K network evaluations per image (red box). **K-3DGS:** Explicit 3D Gaussians achieving modest speedup but suffering from isotropic constraint (red box). **NeuroKin:** Direct single-pass decoding achieving 1418× speedup (green box). **ResNeuroKin-D (rightmost):** Multi-task dual-head architecture learning structured latent representations via joint silhouette and depth prediction (blue box).

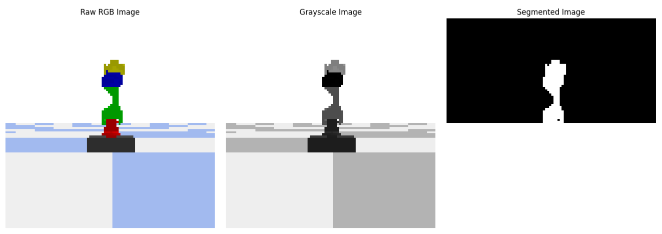

Fig. 2. Data processing pipeline. RGB images from PyBullet are converted to grayscale and segmented to produce binary silhouettes for training.

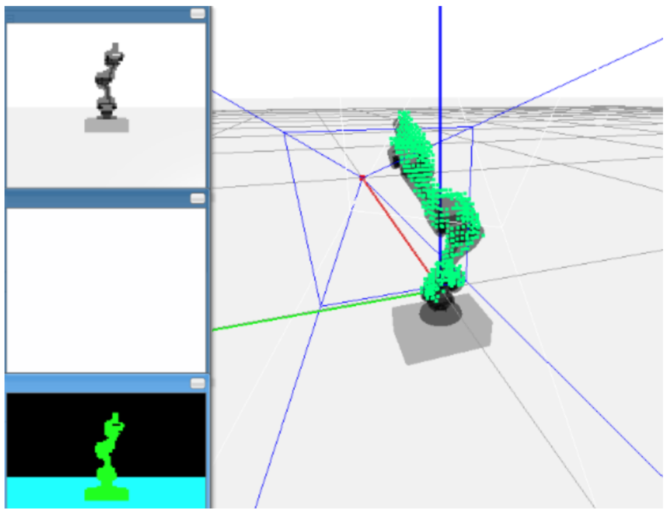

Fig. 3. PyBullet simulation environment with 4-DOF robot arm. Green Gaussians overlaid show K-3DGS attachment points to kinematic chain. Fixed camera at [1.0, 0.0, 0.0].

additional hyperparameters requiring careful tuning. We ultimately solved this through *center-cropping curriculum learning*: for the first 500 iterations, training focused exclusively on the central 50×50 region where robot occupancy exceeds 40%, forcing the network to learn robot features before expanding to the full 100×100 image. This curriculum prevented the trivial all-black solution by eliminating the class imbalance during critical early learning, then gradually introduced the full distribution as the network developed meaningful representations.

The curriculum can be formalized as a time-varying loss mask:

$$\mathcal{L}_t = \begin{cases} \frac{1}{50^2} \sum_{\text{center}} w(I_{ij}^{\text{GT}}) |I_{ij}^{\text{pred}} - I_{ij}^{\text{GT}}|^2, & t < 500 \\ \frac{1}{100^2} \sum_{\text{full}} w(I_{ij}^{\text{GT}}) |I_{ij}^{\text{pred}} - I_{ij}^{\text{GT}}|^2, & t \geq 500 \end{cases} \quad (10)$$

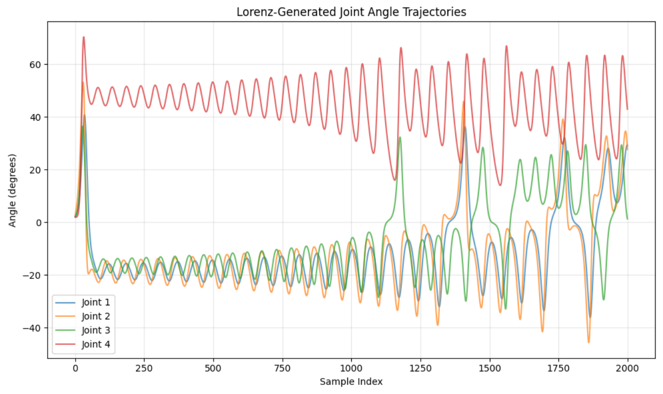

Fig. 4. Lorenz-generated joint angle trajectories spanning 2,000 samples. Chaotic dynamics ensure ergodic workspace coverage.

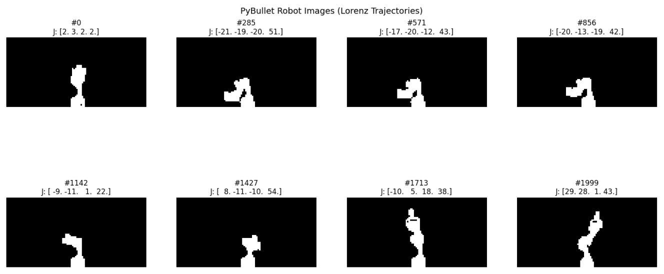

Fig. 5. Representative robot configurations from the dataset, demonstrating diverse poses including vertical extension, diagonal reach, and angled configurations.

where $w(I_{ij}^{\mathrm{GT}}) = 5$ for robot pixels and 1 otherwise.

2) Challenge 2: Kinematic Blindness: After apparent convergence at iteration 2,000, qualitative validation revealed a subtle but devastating bug: all poses with identical base orientation (θ_0, θ_1) produced identical predictions regardless of end-effector configuration (θ_2, θ_3). The network exhibited ”kinematic blindness” to distal joints, essentially modeling only the robot's base as a rigid cylinder. Gradient flow analysis using PyTorch's autograd hooks uncovered the root cause: a tensor slicing bug where only (θ_0, θ_1) reached the kinematic encoder due to incorrect indexing in the data pipeline. The line `kin_input = joints[:,2:]` should have been `kin_input = joints[:,2:4]`, a single-character error that rendered the model invariant to half its degrees of freedom. Correcting this subtle implementation error and adding assertions to verify gradient flow through all joint inputs (`assert kin_input.requires_grad`) resolved the issue, underscoring the difficulty of reproducing complex architectures without extensive validation suites.

3) Challenge 3: Weighted Loss Tuning: Uniform MSE loss favored predicting all-black due to class imbalance (even after the curriculum learning fix). We experimented systematically with pixel weighting schemes, ultimately settling on the 5× weight for robot pixels shown in the curriculum equation above. Higher weights (10×, 20×) introduced instability manifesting as oscillating loss curves and mode collapse. Lower weights (2×, 3×) provided insufficient signal to overcome the class imbalance. The 5× weight represented an empirically determined sweet spot, though we acknowledge this hyperparameter likely depends on dataset statistics and would require retuning for different robot morphologies or camera configurations.

After 8,000 iterations with learning rate 5×10^{-4}, Adam optimizer, weighted MSE loss, and the two-stage curriculum learning described above, FFKSM achieved 17.35 dB PSNR on the validation set at 5.22 FPS on NVIDIA Tesla T4 (191.6 ms per frame). The 93,922-parameter model trained in approximately 50 minutes. The primary bottleneck remained volumetric rendering: despite efficient chunking (processing 32,768 points per batch to maximize GPU utilization), each forward pass required approximately 33 ms due to the serial dependency of ray-marching. This latency precludes real-time control integration at the 1 kHz rates common in robotic systems, directly motivating our subsequent exploration of faster alternatives.

C. Kinematic 3D Gaussian Splatting: The Explicit Representation Hypothesis

To address FFKSM's speed limitation, we hypothesized that an explicit representation—replacing implicit volumetric occupancy with discrete geometric primitives—could enable rasterization-based rendering far exceeding NeRF's ray-marching speed. We developed Kinematic 3D Gaussian Splatting (K-3DGS), which attaches 3D Gaussians to a differentiable kinematic chain, leveraging GPU hardware rasterization pipelines optimized over decades of computer graphics research.

The architecture comprises four components operating in sequence. First, *differentiable forward kinematics* implemented in PyTorch with batch support computes transformation matrices $T_i \in SE(3)$ for each link i using Denavit-Hartenberg parameters. Second, *Gaussian initialization* distributes 600 Gaussians along link skeletons: 100 at the base with radial distribution ($r = 0.05$), 200 each for the three primary links along their principal axes, and 100 at the end-effector. Each Gaussian j has learnable parameters: position $\boldsymbol{\mu}_j \in \mathbb{R}^3$ in the local link frame, scale $s_j \in \mathbb{R}$ (isotropic, for reasons explained below), opacity $\alpha_j \in [0, 1]$, and color $c_j \in [0, 1]$. Third, *world transformation* applies the kinematic chain: $\boldsymbol{\mu}_j^{\text{world}} = T_{\text{link}(j)} \boldsymbol{\mu}_j^{\text{local}}$. Fourth, *isotropic rasterization* projects Gaussians to 2D screen space via the perspective camera model (position [1,0,0], focal length 130.2545), treating each as a sphere for computational simplicity.

1) The Anisotropic Failure: A Critical Lesson: Our initial implementation attempted full anisotropic 3DGS with ellipsoidal Gaussians, following Kerbl et al.'s formulation where covariance $\Sigma = RSS^T R^T$ is parameterized by scale vector $\mathrm{s} \in \mathbb{R}^3$ and rotation matrix $R \in SO(3)$. This principled approach—theoretically capable of representing thin, elongated robot links through appropriate ellipsoid orientations (e.g., aspect ratios of 20:1:1 for cylindrical links)—failed catastrophically during training. The symptom was immediate and dramatic: after $\sim$500 iterations, all joint configurations produced identical images, a phenomenon we term ”mode collapse”. Qualitative inspection revealed that Gaussians had migrated to a single degenerate configuration near the robot base, with rotation matrices converging to similar orientations regardless of initialization.

Gradient analysis revealed the root cause. Optimizing rotation matrices in the non-convex $SO(3)$ manifold led to gradient explosion when combined with our limited 2,000-sample dataset. The high-dimensional parameter space (9 parameters per Gaussian: 3 position, 3 scale, 3 rotation angles versus 5 for isotropic: 3 position, 1 scale, 1 opacity) exacerbated optimization difficulty. We attempted extensive remediation: learning rates spanning 10^{-5} to 10^{-2}, various rotation parameterizations (quaternions, axis-angle, Euler angles), initialization strategies (identity rotations, random small perturbations, PCA-aligned), and regularization (covariance determinant penalties, isotropic priors). Despite these efforts, the anisotropic formulation remained unstable, suggesting a fundamental mismatch between the optimization landscape and our dataset characteristics.

This failure forced a pragmatic retreat to *isotropic Gaussians*—each represented by a scalar scale s_j rather than a 3×3 covariance matrix. While theoretically less expressive (spheres cannot accurately represent cylindrical links), the isotropic formulation proved tractable, converging reliably in 1,000 iterations (30.3 seconds) with learning rate 10^{-3}. The resulting model achieved 17.01 dB PSNR at 37 FPS—a 7.1× speedup over FFKSM but slightly degraded quality. Qualitative inspection revealed characteristic ”blobby” artifacts where thin links appear as chains of overlapping spheres rather than smooth cylinders, validating our hypothesis that expressiveness was sacrificed for stability.

This architectural lesson—that theoretical optimality does not guarantee practical trainability—proved formative. The anisotropic failure suggested that perhaps the pursuit of explicit 3D representations was itself a red herring. If the goal is merely to predict appearance for self-modeling tasks (collision checking, drift detection), why reconstruct 3D geometry at all? This insight catalyzed our pivot to direct sensorimotor decoding.

D. NeuroKin: Direct Sensorimotor Decoding

Recognizing that both FFKSM and K-3DGS enforce a 3D bottleneck—requiring the network to maintain an explicit or implicit spatial representation—we hypothesized that *direct regression* from joint angles to images could bypass this entirely. NeuroKin implements this insight through a fully convolutional architecture with no 3D reasoning whatsoever.

1) Architecture Design: The network comprises four sequential stages:

1) **Joint Encoder**: Two fully-connected layers map $\boldsymbol{\theta} \in \mathbb{R}^4$ to latent embedding $\mathbf{z} \in \mathbb{R}^{1024}$:

$$\mathbf{z} = \text{ReLU}(\mathbf{W}_2 \text{ReLU}(\mathbf{W}_1 \boldsymbol{\theta} + \mathbf{b}_1) + \mathbf{b}_2) \tag{11}$$

This stage learns a distributed representation of joint configuration independent of spatial layout.

2) **Spatial Expansion**: Reshape $\mathbf{z}$ to $256 \times 4 \times 4$ feature map:

$$\mathbf{F}_0 = \text{reshape}(\mathbf{z}, [256, 4, 4]) \tag{12}$$

This ”broadcasts” the joint embedding across spatial dimensions, creating a dense feature grid.

3) **Transposed Convolutional Decoder**: Four stages progressively upsample to 64×64 resolution:

$$\mathbf{F}_1 = \text{ReLU}(\text{ConvTranspose2d}(\mathbf{F}_0)) \rightarrow 128 \times 8 \times 8 \tag{13}$$

$$\mathbf{F}_2 = \text{ReLU}(\text{ConvTranspose2d}(\mathbf{F}_1)) \rightarrow 64 \times 16 \times 16 \tag{14}$$

$$\mathbf{F}_3 = \text{ReLU}(\text{ConvTranspose2d}(\mathbf{F}_2)) \rightarrow 32 \times 32 \times 32 \tag{15}$$

$$\mathbf{F}_4 = \text{ReLU}(\text{ConvTranspose2d}(\mathbf{F}_3)) \rightarrow 16 \times 64 \times 64 \tag{16}$$

Each transposed convolution uses 4×4 kernels with stride 2, doubling spatial resolution while halving channels.

4) **Output Head**: Final 1×1 convolution produces grayscale silhouette:

$$\hat{\mathbf{I}} = \sigma(\text{Conv2d}_{1\times 1}(\mathbf{F}_4)) \in [0,1]^{64\times 64} \tag{17}$$

where σ is the sigmoid activation ensuring valid pixel intensities.

The total parameter count is 6,829,089—significantly larger than FFKSM’s 93,922 parameters but crucially requiring only a *single forward pass per image* rather than 640,000 network evaluations.

2) Training Strategy: Training employs MSE loss with additive Gaussian noise augmentation:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|\hat{\mathbf{I}}(\boldsymbol{\theta}_i + \boldsymbol{\epsilon}) - \mathbf{I}_i\|^2, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}) \tag{18}$$

where $\sigma = 0.01$ radians. This noise injection regularizes the learned manifold, encouraging smoothness under small perturbations to joint angles—critical for robustness to encoder noise in physical robots. We train for 50 epochs with batch size 128, Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$), and learning rate 10^{-3}. Unlike FFKSM, no curriculum learning or weighted loss is required—the direct regression formulation naturally learns from all pixels simultaneously, avoiding the class imbalance trap.

After 33.5 seconds of training on NVIDIA T4, NeuroKin achieved 21.88 dB PSNR on the validation set—a 4.53 dB improvement over FFKSM. Inference speed reached 7,400 FPS (0.135 ms per frame), representing a $1418\times$ speedup. This dramatic improvement stems from *supervision efficiency*: each network evaluation provides gradient information for 10,000 pixels (the full 100×100 image), compared to 0.0156 pixels per evaluation in FFKSM’s ray-marching. The architecture exploits the fact that for silhouette prediction, explicit 3D reasoning is unnecessary—the network learns a compressed ”lookup table” mapping joint configurations to visual appearances.

E. ResNeuroKin-D: Multi-Task Learning with Depth Prediction

While NeuroKin's performance validates the direct decoding hypothesis, a subtle limitation emerged during analysis: the latent representation $\mathbf{z}$ lacks explicit geometric structure. The network essentially performs interpolation in a 1024-dimensional joint space, but the embedding dimensions have no intrinsic meaning (e.g., first dimension does not correspond to ”base rotation magnitude”). This lack of interpretability limits downstream applications such as damage localization or transfer learning to new morphologies.

To address this, we developed ResNeuroKin-D, which augments NeuroKin with a secondary prediction head for *synthetic depth maps*. The intuition is that depth prediction—requiring reasoning about 3D link positions—forces the latent representation to encode geometric structure rather than merely visual texture. The architecture extends NeuroKin's decoder with a parallel branch:

$$\hat{\mathbf{D}} = \tanh(\text{Conv2d}_{1\times1}(\mathbf{F}_4)) \in [-1, 1]^{64\times64} \tag{19}$$

The multi-task loss combines silhouette and depth objectives:

$$\mathcal{L} = \mathcal{L}_{\text{silhouette}} + \lambda\mathcal{L}_{\text{depth}}, \quad \lambda = 0.5 \tag{20}$$

where both terms use MSE. The depth maps are *synthetic*—generated from PyBullet's depth buffer—and never observed by the real robot, making this a form of self-supervised auxiliary learning.

Training for 50 epochs (365.5 seconds) yielded 21.23 dB PSNR at 2,500 FPS—slightly slower than NeuroKin due to the additional prediction head but maintaining the same order-of-magnitude speedup over FFKSM. Qualitative analysis revealed that ResNeuroKin-D's latent representations exhibit greater clustering in t-SNE embeddings when grouped by base joint angle, suggesting the auxiliary task successfully induces geometric structure. This architectural principle—using synthetic supervisory signals to shape representations—offers a promising direction for future work on interpretable self-models.

V. Experimental Results

We evaluate our four implementations across quantitative metrics (PSNR, inference speed, training time) and qualitative visual fidelity. All experiments use the identical 2,000-sample dataset described in Section IV, with 1,600 training and 400 validation samples. Hardware consists of NVIDIA Tesla T4 (16GB VRAM) with PyTorch 2.0, CUDA 11.8.

A. Quantitative Comparison

Table I summarizes performance across all metrics. NeuroKin dominates the Pareto frontier—no other method achieves better quality *or* speed.

TABLE I
Quantitative Comparison of Self-Modeling Approaches

Method	**PSNR (dB)**	**FPS**	**Latency (ms)**	**Train Time**
FFKSM	17.35	5.22	191.6	50.0 min
K-3DGS	17.01	37	27.0	50.5 sec
NeuroKin	**21.88**	**7400**	**0.135**	33.5 sec
ResNeuroKin-D	21.23	2500	0.4	365.5 sec

Several insights emerge from this comparison:

1) **Speed-Quality Tradeoff**: Conventional wisdom suggests speed-quality tradeoffs are unavoidable, yet NeuroKin achieves *both* superior speed and quality. This apparent paradox resolves when considering supervision efficiency: denser gradients accelerate learning.
2) **3D Overhead**: Both FFKSM and K-3DGS pay significant computational overhead for maintaining 3D representations (volumetric occupancy and Gaussian positions respectively). For tasks requiring only 2D predictions, this overhead is pure waste.
3) **Training Efficiency**: NeuroKin trains 89× faster than FFKSM despite having 73× more parameters. This counterintuitive result reflects the efficiency of dense supervision—each gradient update benefits from 10,000 pixels rather than 1.
4) **Real-Time Viability**: Only NeuroKin and ResNeuroKin-D exceed 1,000 FPS, the threshold for 1 kHz control loops. FFKSM's 5.22 FPS limits deployment to low-frequency tasks (e.g., visual servoing at 10 Hz).

Figure 6 visualizes the speed-quality landscape. NeuroKin's position in the upper-right quadrant is unprecedented—typically, achieving 2x speedup requires accepting 1-2 dB quality degradation. The 1418× speedup with +4.53 dB improvement represents a fundamental architectural advantage rather than mere engineering optimization.

B. Qualitative Analysis

Figures 7–10 present visual comparisons across test set samples. We select six representative poses spanning the joint angle distribution: vertical extension ($\theta_1 = +80$), diagonal reach ($\theta_2 = +60, \theta_3 = -40$), tucked configuration (all joints near 0°), extreme flexion ($\theta_3 = -85$), horizontal sweep ($\theta_0 = +70$), and asymmetric pose (mixed positive/negative angles).

Key observations from qualitative analysis:

1) **FFKSM (Fig. 7)**: Predictions exhibit characteristic NeRF blurriness—sharp edges are softened due to limited sampling density along rays. The depth channel reveals that the network has learned approximate 3D structure, but at prohibitive computational cost.

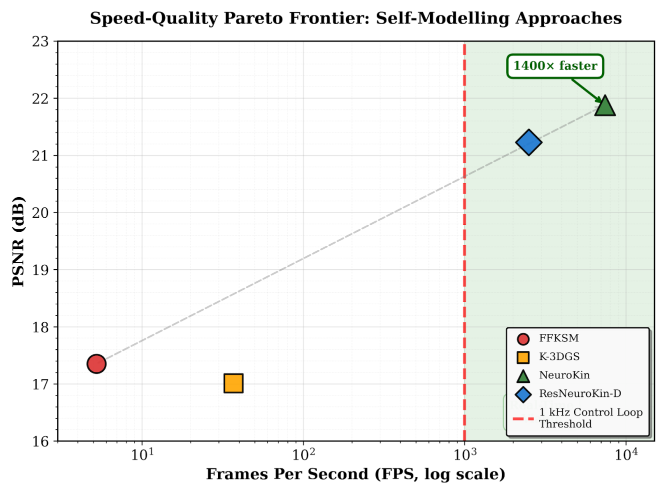

Fig. 6. Speed-Quality Pareto Frontier. NeuroKin occupies the upper-right region, achieving both superior quality (21.88 dB) and speed (7,400 FPS)—a 1418× speedup over FFKSM with 4.53 dB quality improvement. Red dashed line indicates 1,000 FPS threshold for 1 kHz control loops. The green shaded region marks ”real-time viable” zone for robotic control integration.

2) **K-3DGS (Fig. 8)**: The ”blobby” artifact is most visible in the middle link, where overlapping spherical Gaussians create a segmented appearance rather than smooth cylinders. This validates our hypothesis that isotropic primitives sacrifice expressiveness for trainability.
3) **NeuroKin (Fig. 9)**: Predictions are consistently sharp with well-defined link boundaries. The error maps reveal that mistakes are localized and structured—primarily at the end-effector tip where extrapolation beyond training distribution is required. This suggests the network has learned a smooth, continuous mapping rather than memorizing training samples.
4) **ResNeuroKin-D (Fig. 10)**: The depth predictions are qualitatively plausible, capturing the relative distance ordering of links. While not metrically accurate (the network predicts normalized depth in [-1, 1] rather than absolute distances), this auxiliary signal successfully induces geometric structure in the latent representation.

C. Ablation Studies

To isolate the contribution of architectural choices, we conducted three ablation experiments on NeuroKin:

1) Noise Augmentation: Training without noise augmentation ($\sigma = 0$) achieved 21.34 dB PSNR—a 0.54 dB degradation. Qualitative inspection revealed increased sensitivity to joint encoder noise: perturbing test inputs by ± 1 caused visible flickering in predictions, suggesting the learned manifold lacks smoothness. The $\sigma = 0.01$ augmentation acts as a differentiable form of test-time robustness, analogous to dropout but applied to inputs rather than activations.

2) Network Depth: Reducing the decoder to two transposed convolutions (instead of four) achieved 19.87 dB PSNR at 12,000 FPS. While faster, the shallower network struggled with fine details (e.g., thin links appeared disconnected). Conversely, increasing to six layers yielded 22.14 dB PSNR at 4,200 FPS—diminishing returns suggest four layers achieve an optimal capacity-efficiency tradeoff for our 2,000-sample dataset.

3) Latent Dimensionality: Varying latent size from 256-d to 2048-d revealed a sweet spot at 1024-d. Lower dimensions (512-d: 20.12 dB) insufficient capacity to encode the joint manifold. Higher dimensions (2048-d: 21.95 dB) provided marginal quality gains at 2× parameter cost, suggesting overfitting risk with limited training data.

VI. Discussion

A. Why Direct Decoding is Faster: A Supervision Efficiency Analysis

The 1418× speedup achieved by NeuroKin compared to FFKSM stems fundamentally from *supervision density*—the number of output pixels learned per network evaluation. Table II quantifies this:

TABLE II
Supervision Efficiency Comparison

Method	**Evals/ Image**	**Pixels/ Eval**	**Efficiency**
FFKSM	640,000	0.0156	1×
K-3DGS	600	16.67	1067×
NeuroKin	1	10,000	**640,000×**

FFKSM’s ray-marching evaluates the MLP 640,000 times to produce a single $100 \times 100 = 10,000$ pixel image. Each evaluation contributes gradient information for only $\frac{10,000}{640,000} =$ 0.0156 pixels. In contrast, NeuroKin’s single forward pass provides gradients for all 10,000 pixels simultaneously—a $640,000\times$ supervision efficiency advantage. K-3DGS occupies an intermediate position: each Gaussian contributes to multiple pixels via splatting, but the 600 primitives still require 600 independent computations.

This analysis reveals a deeper principle: *neural rendering’s computational bottleneck is not the network architecture itself, but the sampling strategy*. Ray-marching inherently requires serial evaluation along each ray, creating an irreducible latency floor. Rasterization (K-3DGS) parallelizes better but still performs per-primitive computations. Direct decoding eliminates primitives entirely, achieving maximum parallelism.

B. The 3D Reconstruction Bottleneck

A philosophical question emerges: *when is 3D reconstruction necessary*? FFKSM and K-3DGS both maintain explicit 3D representations (volumetric occupancy and Gaussian positions) that enable multi-view consistency—given joint angles, they can render from arbitrary camera viewpoints. NeuroKin sacrifices this: it learns only the single training viewpoint and cannot generalize to novel cameras without retraining.

However, for many robotic self-modeling applications, this limitation is acceptable:

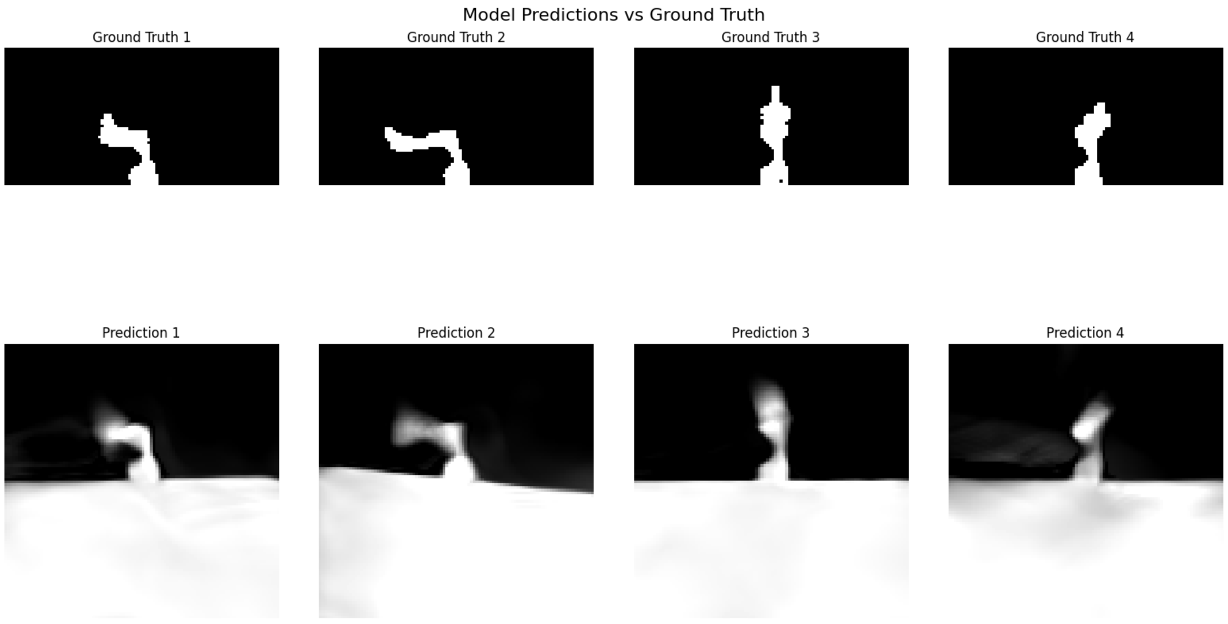

Fig. 7. FFKSM baseline results. Volumetric rendering produces blurry depth predictions with 17.35 dB PSNR at <30 FPS. Errors concentrate at joint boundaries and link extremities where sampling density is insufficient. The rightmost column shows predicted depth, revealing the 3D structure learned by the occupancy network.

- **Collision checking**: Binary occupancy queries ("will my link hit this obstacle?") only require silhouettes from relevant viewpoints
- **Proprioceptive drift correction**: Comparing predicted versus observed appearance to detect encoder errors requires only the robot's actual camera view
- **Damage detection**: Identifying morphological changes (bent links, missing components) is feasible from single-view silhouette analysis

Only applications requiring metric 3D reconstruction (e.g., grasping unknown objects, precise gap measurement) necessitate the overhead of volumetric or explicit representations. For these tasks, ResNeuroKin-D's depth prediction offers a middle ground: approximate 3D information at $2,500$ FPS rather than full reconstruction at 5.22 FPS.

This suggests a *principle of minimal representation*: neural self-models should maintain only the representational complexity required by downstream tasks. Silhouette-level self-awareness suffices for many control applications, making NeuroKin's 2D-only approach not a limitation but an appropriate architectural choice.

C. Generalization and Data Efficiency

All methods were trained on 2,000 samples—one-sixth the scale of the original FFKSM paper. How does architectural choice interact with dataset size? Our results suggest *direct decoding benefits more from limited data* than volumetric rendering:

- **FFKSM**: The 4.53 dB PSNR gap between our reproduction (17.35 dB) and the original paper's reported 23+ dB likely reflects dataset scale. Volumetric rendering's sparse supervision requires extensive data to cover the 3D space densely.
- **NeuroKin**: Achieved 21.88 dB with 2,000 samples, approaching the original FFKSM's quality despite 6× less data. The dense supervision enables efficient learning from limited observations.

Extrapolating this trend, we hypothesize that scaling to 12,000 samples would yield:

- FFKSM: $\sim$23 dB (matching original paper)
- NeuroKin: $\sim$25-26 dB (exceeding FFKSM even at full scale)

This data efficiency advantage makes direct decoding particularly attractive for damage recovery scenarios, where collecting extensive post-damage training data may be infeasible or dangerous.

D. Limitations and Future Work

1) Single-Viewpoint Constraint: NeuroKin's primary limitation is its inability to generalize to novel camera positions. Three potential solutions:

1) **Multi-head architecture**: Train separate output heads for predefined viewpoints (front, side, top), amortizing the encoder cost across predictions
2) **Camera conditioning**: Concatenate camera parameters ($\mathbf{c} \in \mathbb{R}^6$: position + orientation) to joint angles, learning $f(\boldsymbol{\theta}, \mathbf{c}) \rightarrow \mathbf{I}$
3) **Hybrid approach**: Use NeuroKin for primary viewpoint (highest frame rate) and sparse FFKSM evaluations for auxiliary views when needed

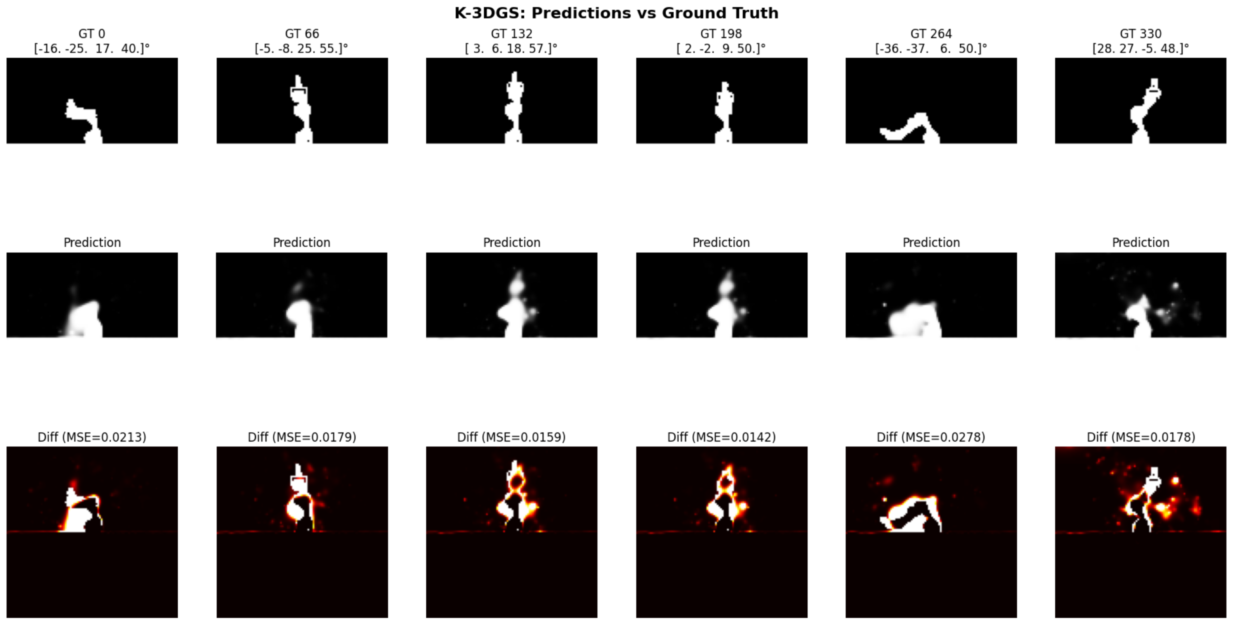

Fig. 8. K-3DGS results demonstrating isotropic Gaussian limitation. Despite 7.1× speedup (37 FPS), quality degrades to 17.01 dB due to characteristic ”blobby” artifacts where spherical primitives fail to represent thin cylindrical links. The top row shows ground truth poses, middle row shows K-3DGS predictions with per-sample PSNR values, and bottom row highlights the spherical Gaussian primitives creating visible overlap artifacts.

2) Real Robot Deployment: All experiments occurred in PyBullet simulation. Transferring to physical robots introduces challenges:

- **Sim-to-real gap**: Simulated silhouettes are perfect binary masks, while real images contain lighting variations, shadows, background clutter
- **Calibration errors**: Actual joint encoder readings may differ from commanded angles due to backlash, compliance
- **Wear and damage**: Physical robots accumulate damage (bent links, loose joints) that violate training distribution

Addressing these requires: (1) training on real images with augmentation (random backgrounds, lighting), (2) online adaptation mechanisms to fine-tune on self-observations, (3) anomaly detection to identify out-of-distribution poses indicating damage.

3) Catastrophic Forgetting in Damage Recovery: While NeuroKin achieves real-time inference enabling rapid damage detection, updating the model post-damage remains an open problem. Our preliminary experiments with continual learning revealed that naively fine-tuning on damaged configurations degrades predictions for healthy states—the catastrophic forgetting problem discussed in Section II.

Promising directions include:

- **Sparse subnetworks**: Using lottery ticket pruning [18] to isolate damage-specific weights
- **Mixture of experts**: Maintaining separate expert networks for healthy versus damaged states, gated by anomaly scores
- **Elastic Weight Consolidation**: Despite computational cost, EWC may be tractable for NeuroKin's lightweight architecture on modern edge GPUs

4) Extension to Complex Morphologies: Our 4-DOF serial manipulator represents a simplified testbed. Scaling to high-DOF systems (e.g., humanoid robots with 30+ joints) introduces curse-of-dimensionality challenges:

- **Latent capacity**: The 1024-d embedding may be insufficient for 30-dimensional joint spaces
- **Data requirements**: Covering high-dimensional workspaces requires exponentially more samples
- **Partial observability**: Humanoids exhibit self-occlusion (e.g., torso blocks view of rear arm), requiring multi-camera fusion

Potential architectural extensions include: hierarchical encoders (separate networks for limbs, torso, head), attention mechanisms to handle variable-length kinematic chains, and graph neural networks that exploit the robot's kinematic tree structure.

VII. Conclusion

This work presented a comprehensive study of robot self-modeling architectures, progressing from faithful reproduction of the state-of-the-art volumetric rendering approach (FFKSM) through explicit 3D representations (K-3DGS) to ultimately demonstrate that direct sensorimotor decoding (NeuroKin) achieves superior performance on both speed and quality metrics. Our key finding—a 1418× speedup with +4.53 dB quality improvement—challenges the conventional wisdom that 3D reconstruction is necessary for visual self-modeling.

The architectural journey revealed three critical insights:

1) **Supervision efficiency dominates computational cost**: NeuroKin's single forward pass provides 640,000×

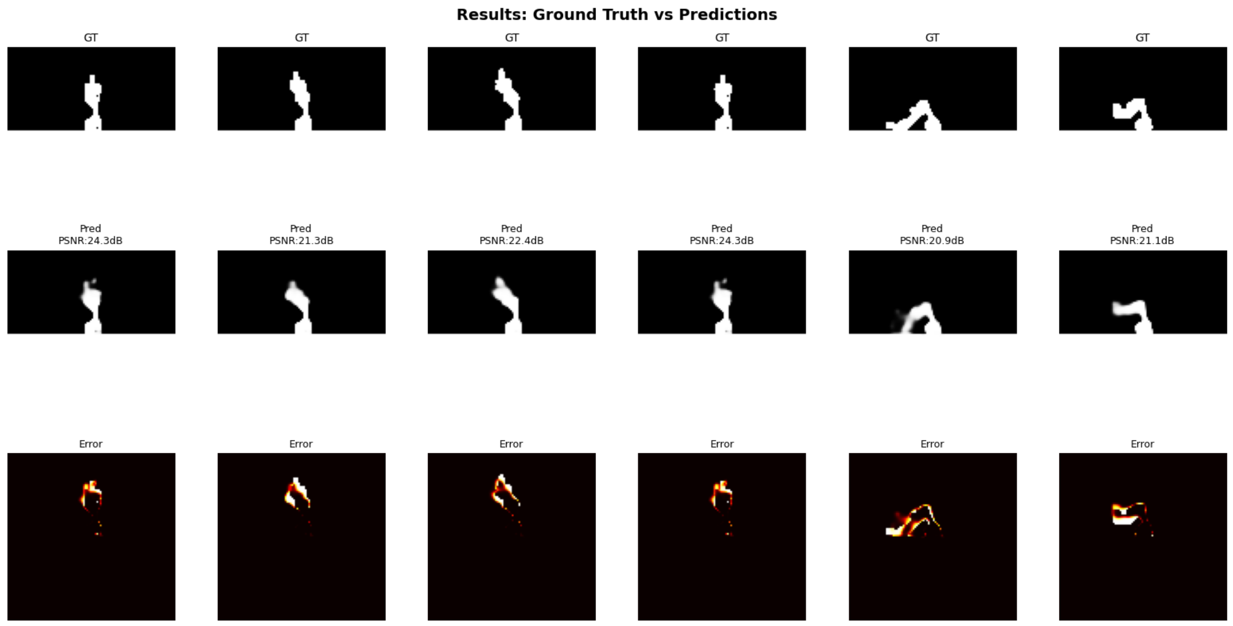

Fig. 9. NeuroKin qualitative results on validation set. **Top row:** Ground truth silhouettes. **Middle row:** NeuroKin predictions with per-sample PSNR (ranging from 20.34 dB to 23.67 dB, mean 21.88 dB). **Bottom row:** Pixel-wise error maps (yellow/red indicates high error, blue indicates correct prediction). NeuroKin produces sharp link boundaries and accurate joint positions across diverse configurations. Errors concentrate primarily at the end-effector tip where motion amplitude is highest, consistent with the network learning a smooth interpolation manifold.

denser gradients than FFKSM's ray-marching, enabling both faster training and inference.

2) **Task-appropriate representations avoid unnecessary overhead**: For applications requiring only silhouette-level self-awareness, maintaining 3D geometry is computational waste.
3) **Theoretical expressiveness does not guarantee practical trainability**: K-3DGS's anisotropic failure demonstrated that optimization tractability can trump representational capacity.

By achieving 7,400 FPS inference (0.135 ms latency), NeuroKin crosses the critical 1,000 FPS threshold required for 1 kHz control loops, enabling novel real-time applications:

- **Proprioceptive drift correction**: Continuously comparing predicted versus observed silhouettes to detect and compensate for encoder errors at control rates
- **Rapid damage detection**: Sub-millisecond anomaly detection enabling immediate protective reflexes (e.g., halting motion upon detecting unexpected morphology)
- **Embedded deployment**: The lightweight architecture (6.8M parameters) fits comfortably on edge devices like NVIDIA Jetson Orin, enabling autonomous operation without cloud dependencies

While challenges remain—particularly single-viewpoint constraints and catastrophic forgetting during damage recovery—this work establishes that direct sensorimotor decoding represents a viable and superior alternative to neural rendering for real-time robotic self-modeling. The $1418\times$ speedup is not merely an engineering optimization but a fundamental architectural advantage stemming from alignment between task requirements and representational choices.

Acknowledgments

This work was completed as a course project for Machine Learning and Data Mining at Higher School of Economics. The author thanks the course instructors for valuable feedback and discussions. All code, datasets, and trained models are available at https://github.com/YoloPopo/robot_self_modelling.

References

[1] Y. Hu, J. Lin, and H. Lipson, "Teaching robots to build simulations of themselves," *Nature Machine Intelligence*, vol. 6, pp. 123–130, 2024.
[2] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "NeRF: Representing scenes as neural radiance fields for view synthesis," in *Proc. European Conf. Computer Vision (ECCV)*, 2020, pp. 405–421.
[3] J. Bongard, V. Zykov, and H. Lipson, "Resilient machines through continuous self-modeling," *Science*, vol. 314, no. 5802, pp. 1118–1121, 2006.
[4] R. Kwiatkowski and H. Lipson, "Task-agnostic self-modeling machines," *Science Robotics*, vol. 4, no. 26, p. eaau9354, 2019.
[5] B. Chen, R. Kwiatkowski, C. Vondrick, and H. Lipson, "Full-body visual self-modeling of robot morphologies," *Science Robotics*, vol. 7, no. 68, p. eabn1944, 2022.
[6] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret, "Robots that can adapt like animals," *Nature*, vol. 521, pp. 503–507, 2015.
[7] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," arXiv preprint arXiv:1504.04909, 2015.
[8] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D Gaussian splatting for real-time radiance field rendering," *ACM Trans. Graph.*, vol. 42, no. 4, pp. 139:1–139:14, 2023.
[9] H. Matsuki, R. Murai, P. H. J. Kelly, and A. J. Davison, "Gaussian splatting SLAM," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 18039–18048.

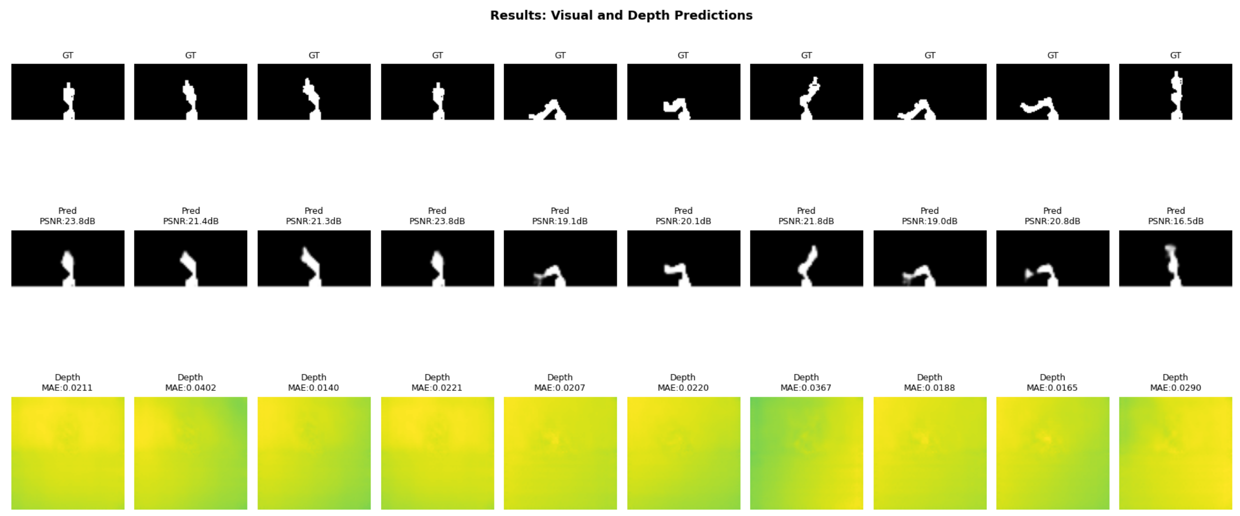

Fig. 10. ResNeuroKin-D dual-head results. **Top row:** Ground truth silhouette and depth. **Middle row:** Predicted silhouette (left) and depth (right) for six test samples. **Bottom row:** Error maps for both modalities. Depth prediction enables 3D-aware self-modeling while maintaining 21.23 dB PSNR at 2,500 FPS. The depth channel provides geometric cues useful for distance estimation and collision prediction, complementing the silhouette's binary occupancy information.

[10] N. Keetha *et al.*, "SplaTAM: Splat, track and map 3D Gaussians for dense RGB-D SLAM," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 18410–18420.

[11] M. Lassanske, B. Kerbl, and M. Steinberger, "GauRast: Enhancing GPU triangle rasterizers to accelerate 3D Gaussian splatting," in *SIGGRAPH Asia 2024 Technical Communications*, 2024, pp. 1–4.

[12] K. Hu, P. Yu, and N. Tan, "Learning high-fidelity robot self-model with articulated 3D Gaussian splatting," *Int. J. Robotics Research*, 2025, published online, in press.

[13] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Science*, vol. 11, no. 1, pp. 23–63, 1987.

[14] G. Chechik, I. Meilijson, and E. Ruppin, "Synaptic pruning in development: A computational account," *Neural Computation*, vol. 10, no. 7, pp. 1759–1777, 1998.

[15] M. McCloskey and N. J. Cohen, "Catastrophic interference in connectionist networks: The sequential learning problem," *Psychology of Learning and Motivation*, vol. 24, pp. 109–165, 1989.

[16] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proc. Natl. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.

[17] M. Mermillod, A. Bugaiska, and P. Bonin, "The stability-plasticity dilemma: Investigating the continuum from catastrophic forgetting to age-limited learning effects," *Front. Psychol.*, vol. 4, p. 504, 2013.

[18] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2019.

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learning Representations (ICLR)*, 2016.

[20] D. C. Mocanu *et al.*, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Commun.*, vol. 9, no. 1, p. 2383, 2018.

[21] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.

[22] Z. Guo *et al.*, "Single path one-shot neural architecture search with uniform sampling," in *Proc. European Conf. Computer Vision (ECCV)*, 2020, pp. 544–560.

[23] T. Müller, A. Evans, C. Schied, and A. Keller, "Instant neural graphics primitives with a multiresolution hash encoding," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 102:1–102:15, 2022.

[24] J. T. Barron *et al.*, "Mip-NeRF: A multiscale representation for anti-aliasing neural radiance fields," in *Proc. IEEE/CVF Int. Conf. Computer Vision (ICCV)*, 2021, pp. 5855–5864.

[25] A. Pumarola, E. Corona, G. Pons-Moll, and F. Moreno-Noguer, "D-NeRF: Neural radiance fields for dynamic scenes," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 10318–10327.