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

December 17, 2025

Course: Machine Learning and Data Mining Implementation Project

The Reality Gap: Why Self-Modeling?

1. **The Problem:** Robots in unstructured environments (Space, Rescue) suffer damage.

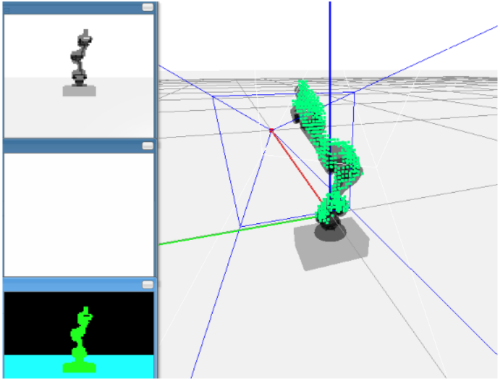


Figure 1: 4-DOF Manipulator in PyBullet Simulation Environment.

The Reality Gap: Why Self-Modeling?

1. **The Problem:** Robots in unstructured environments (Space, Rescue) suffer damage.
2. **The Failure Mode:** Traditional controllers use fixed kinematic files (URDF). If the body bends, the model drifts $\rightarrow$ **Catastrophic Failure**.

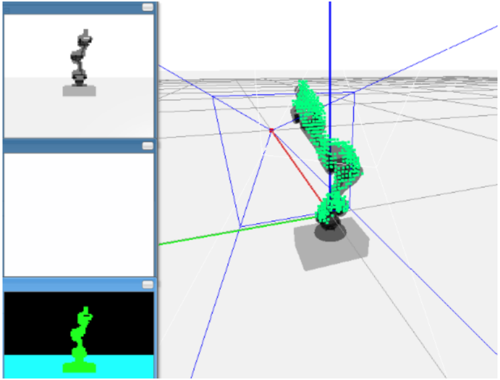


Figure 1: 4-DOF Manipulator in PyBullet Simulation Environment.

The Reality Gap: Why Self-Modeling?

1. **The Problem:** Robots in unstructured environments (Space, Rescue) suffer damage.
2. **The Failure Mode:** Traditional controllers use fixed kinematic files (URDF). If the body bends, the model drifts $\rightarrow$ **Catastrophic Failure**.
3. **The Goal:** A robot that learns its own shape from vision in real-time.

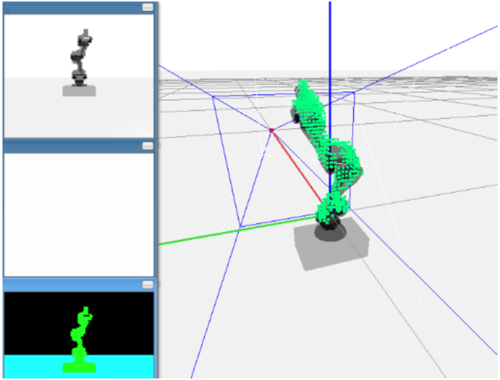


Figure 1: 4-DOF Manipulator in PyBullet Simulation Environment.

The Reality Gap: Why Self-Modeling?

1. **The Problem:** Robots in unstructured environments (Space, Rescue) suffer damage.
2. **The Failure Mode:** Traditional controllers use fixed kinematic files (URDF). If the body bends, the model drifts → **Catastrophic Failure**.
3. **The Goal:** A robot that learns its own shape from vision in real-time.
4. **The Bottleneck:**

Current SOTA - FFKSM Nature 2024

Uses Neural Radiance Fields (NeRF).
Latency: ~5.22 FPS (192ms).
Required: >1000 FPS (1ms) for control loops.

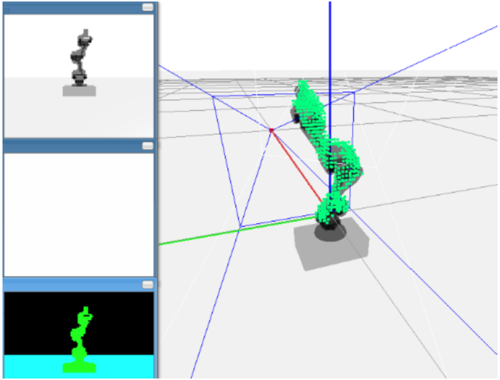


Figure 1: 4-DOF Manipulator in PyBullet Simulation Environment.

Project Contributions & Scope

1. Reproduction

FFKSM (Baseline)

Implemented from scratch.
Solved "Kinematic Blindness" Class Imbalance bugs.

Result: 5.22 FPS

2. Failed Hypothesis

K-3DGS (Explicit)

Tried Kinematic 3D Gaussians.
Failed anisotropic optimization.

Result: "Blobby" Artifacts

3. Innovation

NeuroKin (Ours)

Direct Sensorimotor Decoding.
Bypassed 3D reconstruction entirely.

Result: 7,400 FPS

Data Generation via Chaotic Dynamics

Problem: Random motor babbling produces jerky, discontinuous motion that fails to capture kinematic dependencies.
Solution: We employ **Lorenz Attractors** to generate smooth, ergodic trajectories that efficiently explore the workspace.

$$
\begin{aligned}
\dot{x} &= \sigma(y - x) \\
\dot{y} &= x(\rho - z) - y \\
\dot{z} &= xy - \beta z
\end{aligned}
$$

Parameters: $\sigma = 10, \rho = 28, \beta = 8/3$.

Generates deterministic but non-repeating joint angles for 4-DOF.

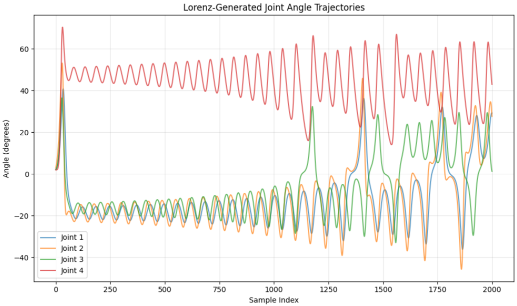


Figure 2: Chaotic trajectories (2,000 samples) ensuring ergodic workspace coverage.

Data Processing Pipeline

We constructed a robust pipeline to convert simulation output into training tensors.

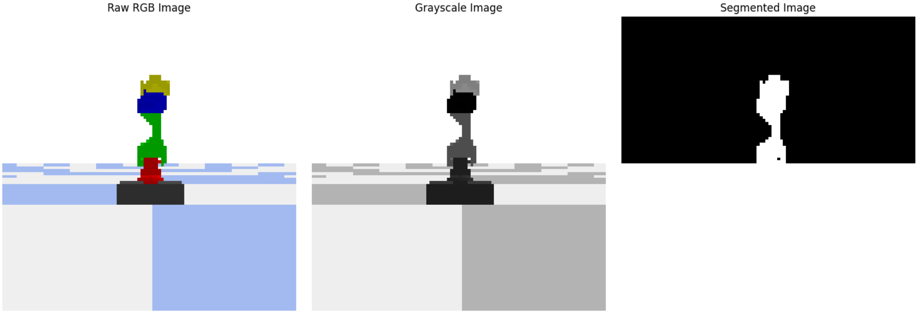


Figure 3: Data preprocessing pipeline converting RGB input to binary training masks.

Input	**Processing**	**Output**
PyBullet RGB ($H \times W \times 3$)	Thresholding (>240) & Grayscale	Binary Masks (100×100)

Dataset Scale: 2,000 samples (1,600 Train / 400 Val). This constrained regime tests data efficiency.

Method 1: FFKSM Architecture (Reproduction)

Approach: Volumetric Querying. Condition Density σ on Joints θ.

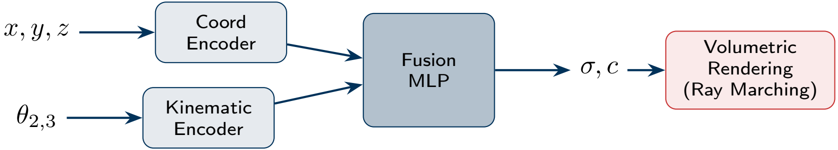


Figure 4: FFKSM architecture with Split-Encoders and Volumetric Rendering head.

- **Split Encoder:** Decouples kinematics from geometry.
- **Bottleneck:** Ray Marching = 640k evals/image.

Results

PSNR: 17.35 dB
Speed: 5.22 FPS (Too slow)

Engineering Challenge: The "Black Screen" Convergence

The Pathology:

- Robot occupies only $\sim$15% of pixels.
- Network converged to **all-zeros** (local minima, Loss $\approx$ 0.15).

The Solution: Curriculum Learning

- **Phase 1 ($t < 500$):** Train ONLY on center 50×50 crop (Occupancy $>$ 40%).
- **Phase 2 ($t \geq 500$):** Expand to full image.

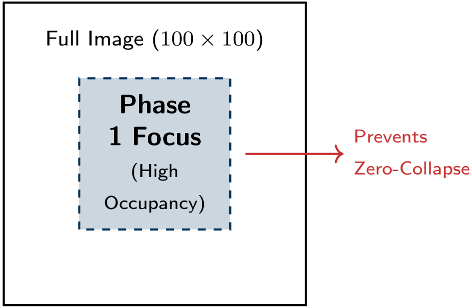


Figure 5: Curriculum Learning Strategy.

Method 2: K-3DGS Architecture (Failed Hypothesis)

Hypothesis: Rasterization is faster than Ray-Marching.

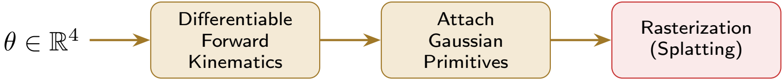


Figure 6: Kinematic 3D Gaussian Splatting (K-3DGS) pipeline.

Optimization Failure

Anisotropic: Too slow and unstable.
Isotropic: Stable, but "blobby".

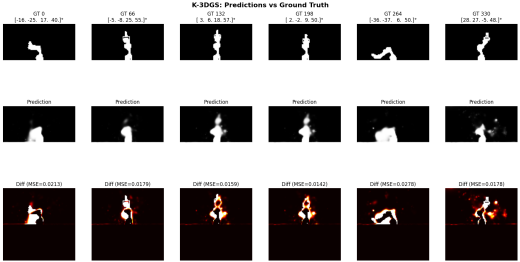


Figure 7: "Blobby" artifacts due to isotropic constraints (17.01 dB).

The Pivot: Bypassing 3D reconstruction for Direct Sensorimotor Decoding.

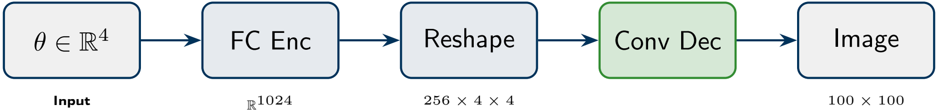


Figure 8: NeuroKin Direct Decoding Architecture.

Why it wins:

- **O(1) Complexity:** Single pass.
- **Dense Supervision:** 10,000 pixels updated per eval.

Performance

Speed: 7,400 FPS (1400x faster)
Quality: 21.88 dB (+4.53 dB)

Method 4: ResNeuroKin-D (Multi-Task Extension)

Motivation: Pure NeuroKin lacks geometric structure in the latent space. **Solution:** Force the network to learn *Synthetic Heuristic Depth* alongside silhouettes.

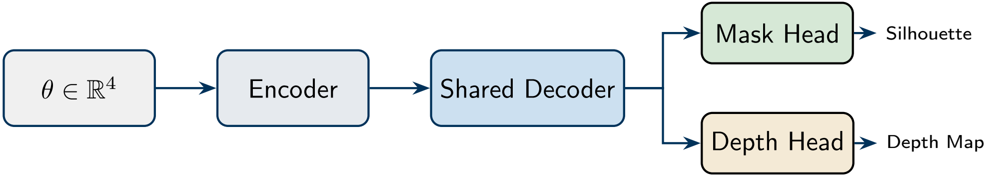


Figure 9: ResNeuroKin-D Dual-Head Architecture.

- **Geometric Prior:** Depth loss forces spatial awareness.
- **Trade-off:** 2,500 FPS (Slower than NeuroKin, but still > 1 kHz).
- **Heuristic Depth:** Generated via kinematic distance.

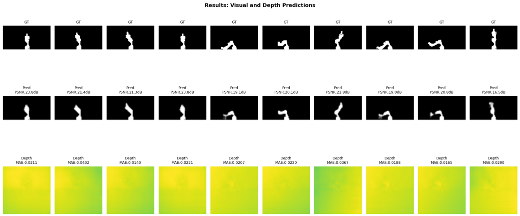


Figure 10: Dual output predictions (Silhouette + Depth).

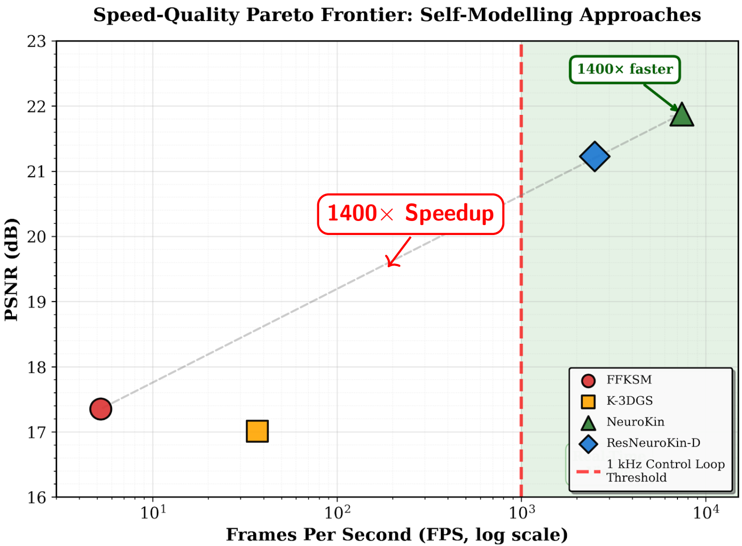
Quantitative Comparison of Self-Modeling Approaches

Table 1: Quantitative comparison of self-modeling architectures.

Method	PSNR	FPS	Latency	Train Time	Speedup
FFKSM	17.35 dB	5.22	191.6 ms	50.0 min	1.0×
K-3DGS	17.01 dB	37	27.0 ms	0.8 min	7.1×
NeuroKin (Ours)	**21.88 dB**	**7400**	**0.135 ms**	**0.5 min**	**1418**×
ResNeuroKin-D	21.23 dB	2500	0.4 ms	6.0 min	479×

- **NeuroKin** achieves a massive **1418x speedup** over the baseline.
- **FFKSM** latency (192ms) is unusable for real-time control.
- **ResNeuroKin-D** maintains >400x speedup while learning depth.

The Pareto Frontier: Speed vs Quality



Key Result

NeuroKin dominates:

- **7,400 FPS**
- **+4.53 dB PSNR**

Ablation: Data Efficiency (RQ3)

Question: How much data does NeuroKin actually need? **Answer:** NeuroKin beats FFKSM with only **25%** of the data.

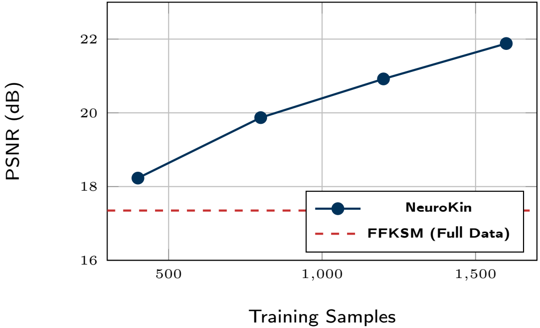


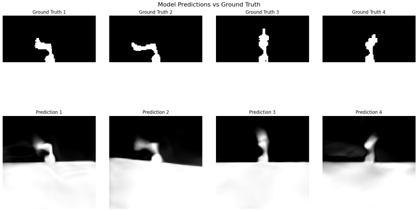
Figure 12: Performance vs Dataset Size.

Key Insight

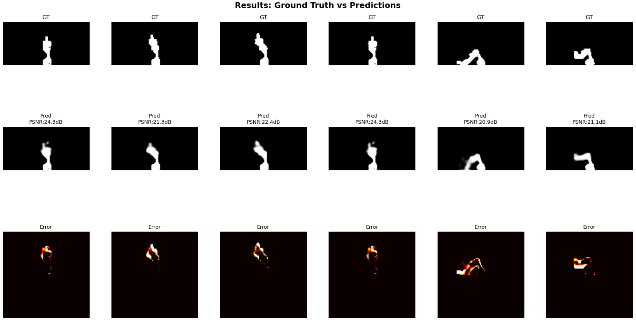
With just **400 samples**, NeuroKin achieves **18.23 dB**, exceeding FFKSM's performance on the full 1,600 dataset.

This confirms that direct decoding is significantly more **sample efficient** than volumetric rendering.

Qualitative Comparison



(a) FFKSM (Baseline): Blurred Boundaries.

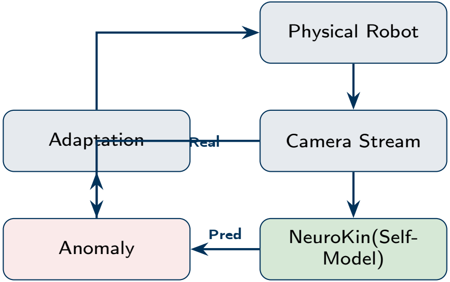


(b) NeuroKin (Ours): Sharp Edges.

Figure 13: Qualitative comparison of generated robot silhouettes.

Future Directions: Towards Resilient Autonomy

- **Multi-View Generalization:**
  - Current limitation: Single fixed camera.
  - Proposal: Train on **multi-view datasets** to learn a view-invariant latent representation.
- **Evolving Sparse Topologies:**
  - Use **Evolutionary Algorithms (NEAT)** to prune the NeuroKin network.
  - Goal: Find the "minimal viable brain" for microcontrollers.
- **Real-Time Damage Adaptation Loop:**
  - Close the control loop: Use the $<$1ms inference to detect damage and adapt online.



Conclusion & Impact

1. **Rigorous Reproduction:** Validated FFKSM, exposed latency floor (5.22 FPS).
2. **Negative Result:** Kinematic 3DGS is unstable without anisotropic constraints.
3. **SOTA Performance: NeuroKin** (7,400 FPS).

Why it matters?

By discarding 3D reconstruction, we enable **1 kHz control loops** on edge hardware (Jetson Orin), allowing robots to detect damage in < 1 ms.

Code available: `https://github.com/YoloPopo/robot_self_modelling`

Thank You!

Questions?

Muhammad Zeeshan Asghar
Higher School of Economics