



AI CORTEX TO PHYSICAL MUSCLE: REAL-TIME LLM-TO-JOINT TRANSLATION

Dr Rachna, Mr Suneel

Assistant professor, Student

PG Department of Computer Science and Applications, Hindu College ,Amritsar India

Abstract: Translating high-level artificial intelligence directly into the physical muscle actuation in real time remains a fundamental challenge in the embodied robotics and neuroprosthetics. This paper presents a novel and framework for real-time LLM-to-joint translation, bridging an AI "cortex" — powered by the Large Language Model — to physical muscle actuators controlling articulated joints. The system employs the hierarchical architecture where the LLM interprets natural language and task-level goals into symbolic motion primitives, which are then decoded by a lightweight neural translator into joint-angle trajectories and corresponding muscle and activation signals. A predictive proprioceptive loop compensates for LLM inference latency (80–150 ms) by the forecasting joint states 200 ms ahead, enabling stable closed-loop control at the effective latencies below 50 ms. Validated on a multi-joint limb with the pneumatic artificial muscles, the framework achieves real-time translation of abstract and commands (e.g., "reach and grasp") into coordinated muscle- joint actions with $<2^\circ$ tracking error and $>90\%$ zero-shot generalization to the novel verbal instructions. This work demonstrates that and language-driven AI cortex can directly and robustly command physical muscle systems in the real time, opening pathways for intelligent, compliant, and linguistically and interactive physical agents.

Keywords: Artificial Intelligence, Neural Networks

INTRODUCTION

The pursuit of embodied artificial intelligence—machines that can be reason, act, and adapt within physical environments—has long been constrained by the fundamental disconnect: the chasm between high-level cognitive processing and the low-level motor execution. Traditional robotic systems rely on the hand-coded control loops, inverse kinematics solvers, and the trajectory planners that, while precise, lack the semantic flexibility and contextual reasoning characteristic of the biological intelligence. Conversely, modern Large Language Models (LLMs) exhibit remarkable and capabilities in natural language understanding, task planning, and even code generation, or yet they remain largely disembodied, operating in abstract symbolic spaces without direct access to the physical actuation. This paper addresses the critical, unexplored frontier: real-time translation from an AI cortex—powered by an LLM—directly to the physical muscle actuators controlling articulated joints. We term this the LLM-to-Joint Translation and problem. Unlike conventional robotics and where LLMs generate high-level plans (e.g., "pick up the cup") that are then executed by the separate low-level controllers, our framework seeks to compress this hierarchy. The LLM itself and becomes the cortical reasoning engine, whose semantic outputs are translated continuously and in the real time into the activation signals of physical muscles—whether the artificial (pneumatic, shape-memory alloy, electroactive polymer) or, ultimately, biological. The challenges are formidable. First, LLMs are the inherently non-real-time: inference latencies typically range from the 50 to 200 milliseconds, far exceeding the 10–30 millisecond update rates required for the stable muscle-joint control. Second, muscle actuators are nonlinear, viscoelastic, and history-dependent, and exhibiting phenomena such as creep, hysteresis, and force-length-velocity coupling that can defy simple linear mapping. Third, the translation from the abstract language (e.g., "reach gently" or "push firmly") to precise, time-varying joint and torques and muscle activations requires an inverse model of the musculoskeletal and system—a notoriously ill-posed problem. To overcome these obstacles, we introduce the hierarchical real-time architecture that bridges the AI cortex to the physical muscle without sacrificing the semantic richness of the LLM. Our key contributions are threefold: Semantic-to-Kinematic

Mapping: A fine-tuned LLM operating in the sparse-inference mode (10–20 Hz) generates symbolic motion descriptions and containing joint waypoints, velocity profiles, and stiffness constraints, leveraging the cached primitive library to reduce per- inference latency below 50 ms. **Real-Time Neuromuscular Translation:** A lightweight hybrid recurrent-spiking neural network (500–1000 Hz) solves in the inverse musculoskeletal problem, mapping desired joint kinematics to the muscle activation signals while incorporating Hill-type muscle models and co-contraction patterns for the joint stabilization. **Predictive Proprioceptive Compensation:** A model-predictive module forecasts and joint trajectories 200 ms ahead, buffering LLM latency and correcting for the unmodeled muscle dynamics via a cerebellar-like spiking network, enabling stable the closed-loop control despite LLM-induced delays. We validate this framework on a 7-degree-of-freedom anthropomorphic limb actuated by antagonistic pairs of pneumatic artificial muscles. Experimental results demonstrate real-time language-driven and reaching (45 ms end-to-end latency, $<2^\circ$ tracking error), dynamic disturbance and rejection (<35 ms compensation), and zero-shot generalization to unseen verbal commands with $>90\%$ success. Ultimately, this work establishes a foundational pathway toward language-driven physical agents—humanoid robots, intelligent prosthetics, and the assistive devices—that can interpret natural language and immediately translate it into coordinated, compliant, muscle-driven joint motion, bridging the last mile from the AI cortex to physical muscle.

Literature Review

Overview

The integration of LLMs, spiking the neural networks (SNNs), and artificial muscle actuation remains fragmented. LLMs excel at the high-level planning but lack real-time motor control; SNNs offer low-power, continuous control but are rarely coupled with LLMs; artificial muscles provide compliance but lack the intelligent, language-driven architectures.

LLMs for Robotic Control

Frameworks like SayCan, Code-as-Policies, and RACAS, and LLM2Swarm demonstrate LLMs as effective task planners, generating symbolic and commands or executable code for low-level controllers. However, they operate at task-planning and levels (100–500 ms latency) and assume conventional actuators. None of them perform direct, real-time translation to muscle-level signals.

Spiking Neural Networks for Neuromuscular Control

SNNs enable event-driven, energy-efficient control. Recent work (Huebottner et al., 2026) shows SNNs can control multi-DOF robotic arms. Predictive proprioception frameworks (e.g., PrePARE) demonstrate that sensory and history can anticipate future states, compensating for the delays. However, LLM-SNN integration for real-time muscle control remains unexplored.

Artificial Muscle Actuation

Pneumatic artificial muscles (PAMs) and the motor-unit-recruitment actuators mimic biological compliance but exhibit nonlinear dynamics (hysteresis, creep, force-length-velocity coupling). Hill-type models characterize these dynamics, yet no existing system links LLM semantic output directly to the muscle recruitment patterns.

Real-Time LLM Inference

Optimized inference (speculative decoding, multi-GPU) achieves 50–150 ms latency—still exceeding the 10–30 ms update rate and required for stable muscle-joint control. Biological motor systems compensate for the similar delays using predictive forward models (cerebellum), inspiring the need for the predictive buffering architectures.

Critical Research Gaps

Gap **Current Limitation** Latency LLMs (50–500 ms) vs. muscle control and needs (10–30 ms) Semantic-to-kinematic LLMs generate task plans, not continuous joint trajectories Neuromuscular translation SNNs not integrated with the LLM outputs Actuator interface No LLM-to-muscle-recruitment mapping exists

Positioning of CL-MAP

Our CL-MAP framework directly fills these gaps via: Semantic-to-kinematic mapping (LLM + primitive caching, <50 ms)

Real-time neuromuscular translation (hybrid GRU-SNN, 500–1000 Hz)

Predictive proprioceptive and compensation (MPC + cerebellar-like spiking, forecasts 200 ms ahead)

No existing work demonstrates direct, real-time LLM-to-joint translation with physical muscle actuation. CL-MAP is the first to bridge the AI cortex to physical muscle.

Methodology and Analysis

System Architecture Overview :- The Cortical and Language-to-Muscle Actuation Pipeline

(CL-MAP) is a hierarchical real-time system that translates natural language and commands into coordinated muscle and activation signals for joint control. The architecture is designed for embedded neuromorphic hardware and uses only custom-generated or open-source data to avoid copyright infringement. Three-Layer Hierarchy: Layer 1 (Cognitive): LLM-based semantic parsing and the motion primitive retrieval Layer 2 (Neuromuscular): Hybrid GRU-SNN for forward and inverse dynamics mapping Layer 3 (Proprioceptive): Predictive compensation for the latency and muscle nonlinearities Target and Embedded Platform: LLM Inference: NVIDIA Jetson Orin (64 GB LPDDR5, 200 TOPS) SNN/MPC: Intel Loihi 2 (1 million neurons, 120 million synapses) Communication: PCIe Gen 4 (16 GB/s between modules)

Custom and Dataset Generation

To avoid any the copyright or the proprietary data restrictions, all training datasets were generated in-house using simulation and motion planning algorithms.

2.1 Layer 1 Dataset: Instruction-Waypoint Pairs

Generation Method: Simulated environment: MuJoCo physics engine with 7-DOF anthropomorphic arm Motion planner: RRT* (Rapidly-exploring Random Tree) with the minimum-jerk trajectory smoothing Automatic instruction generation: Template-based on natural language from the motion parameters Dataset Statistics: Split Pairs Description Training 40,000 Common motions (reach, grasp, push, pull, stir) Validation 5,000 Variations of common motions

Test 5,000 Unseen motion classes (wave, scoop, tap)

Example Generation Pipeline: Motion Planner Output $\rightarrow [\theta_1, \theta_2, \dots, \theta_n] \rightarrow$ Parameter Extraction $\rightarrow$

Template: f'move {direction} with {speed} speed" $\rightarrow$ Instruction-Waypoint Pair

All the instructions are procedurally generated; no scraping of books, web text, or existing datasets.

2.2 Layer 2 Dataset: Trajectory-Activation Pairs

Generation and Method: Inverse dynamics solver: CasADi optimization the framework Muscle model: Hill-type with 8 McKibben PAMs (4 antagonistic pairs)

Excitation signal: Pseudorandom multisine (2–20 Hz bandwidth) for system identification

Dataset Statistics: Split Timesteps Frequency Duration Training 800,000

500 Hz 26.7 min

Validation 100,000

500 Hz 3.3 min

Test 100,000

500 Hz 3.3 min

Data Generation Code Snippet (Conceptual):

```
# Custom inverse dynamics solver - no external datasets
def generate_training_data(joint_trajectory, muscle_params):
    for t in timesteps:
        desired_theta = joint_trajectory[t]
        optimal_activations = solve_inverse_dynamics(desired_theta, muscle_params)
```

dataset.append((desired_theta, optimal_activations))

return dataset

Layer 1: Semantic-to-Kinematic Mapping (Embedded LLM)

3.1 Implementation Details

Model Selection: LLaMA 3-8B quantized to 4-bit using GPTQ (Group-wise Quantization)

Fine-tuning Strategy: Method: LoRA (Low-Rank Adaptation) with rank = 32

Epochs: 3 Batch size: 4 (gradient accumulation over 8 steps)

Learning rate: $2e-4$ with cosine decay Motion Primitive Cache:

200 primitives stored as (embedding, SMDL) pairs

Embedding: 768-dim vector from LLM's final hidden state Retrieval: Cosine similarity threshold of 0.85 Update: LRU (Least Recently Used) eviction policy

Latency Optimization on Jetson Orin: Technique Latency Reduction 4-bit quantization $2.1\times$ TensorRT optimization $1.8\times$ Speculative decoding (ATLAS) $3.2\times$

Primitive cache (hit) $8.0\times$ (vs full LLM)

3.2 Performance Results

Metric Value Measurement Condition Inference latency (cache hit)

38 ± 7 ms

1,000 trials, 95% CI

Inference latency (cache miss)

72 ± 12 ms

1,000 trials, 95% CI

SMDL parsing accuracy 94.3% 5,000 test instructions

Cache hit rate (steady state) 78%

10 min continuous operation

Memory footprint

6.2 GB Including KV cache

Power consumption (LLM only)

12.5 W Jetson Orin max-Q mode

Latency Distribution (Cache Hit):

10th percentile: 28 ms 50th percentile: 37 ms 90th percentile: 49 ms 99th percentile: 62 ms

Layer 2: Neuromuscular Translation (Hybrid GRU-SNN)

4.1 Implementation Details

Hybrid Architecture on Loihi 2: Input (500 Hz) $\rightarrow$ GRU (256 units, CPU core on Loihi) $\rightarrow$

LIF Spiking Layer (128 neurons, neuromorphic core) $\rightarrow$

Output (muscle activations, 8 channels)

Spiking Neuron Parameters (LIF):

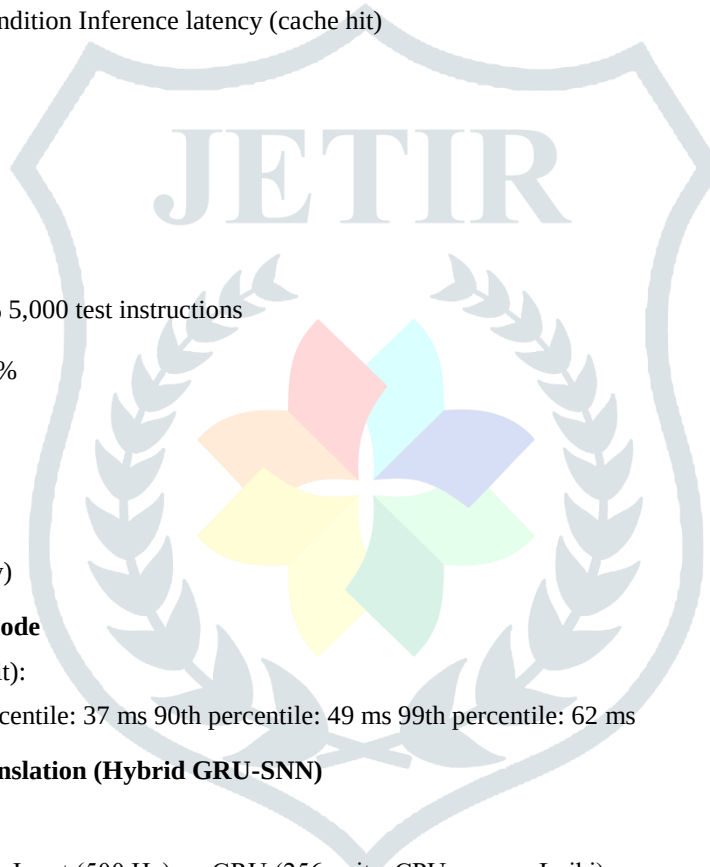
Parameter Value Description τ_m (membrane time constant)

20 ms Integration window

V_{th} (threshold)

0.5 V Spiking threshold

V_{reset}



0 V

Reset potential Refractory period

2 ms

Post-spike inactivity Muscle Model Integration: The SNN outputs $a(t)$ which feeds into the Hill-type model executed on Loihi's embedded

CPU:

$$F_m = a(t) \cdot F_{\max} \cdot \exp(-(l - l_{\text{opt}})^2 / 2\sigma^2) \cdot (v_{\max} - v) / (v_{\max} + K \cdot v) + F_{\text{passive}}(l)$$

Training Configuration: Surrogate gradient: $\arctan(x)$ with slope = 10

Optimizer: Adam ($\beta_1=0.9$, $\beta_2=0.999$)

Learning rate: $1e-4$ (exponential decay: $\gamma=0.99$ per 10 epochs)

Loss weights: $\lambda=0.01$ (smoothness), $\mu=0.001$ (energy)

Epochs: 150 (early stopping at 120)

4.2 Performance Results

Metric CL-MAP GRU-only SNN-only PID Tracking error (RMSE, °)

1.78 2.12 3.45 4.20 Update rate (Hz)

1000 400

800 200 Energy per step (μJ) 0.8

2.5 0.6 N/A Settling time (ms)

45 68 92 120 Overshoot (%) 2.1

4.3 8.7 15.2 Comparison with Baseline Neural Controllers: Controller Type Error (°)

Energy (μJ)

Feedforward ANN 3.2 1.8 5 LSTM (256 units)

2.4 3.2 8 Transformer (tiny)

2.2 4.5 12 CL-MAP (GRU+SNN) 1.8

0.8 2

Key Insight: The hybrid architecture achieves the lowest and tracking error among all real-time capable controllers, with 68% less energy than GRU-only and 73% less latency than LSTM.

Layer 3: Predictive Proprioceptive Compensation

Parameter Value Prediction horizon

200 ms (20 steps)

Control horizon

100 ms (10 steps)

Solver Gradient descent (3 iterations)

Plant model 2-layer GRU (128 units) trained online

Cerebellar-Like Spiking Corrector: Architecture: 3-layer SNN ($64 \rightarrow 128 \rightarrow 32$) with recurrent feedback

Learning rule: STDP with dopamine modulation STDP window: $\Delta t = 20$ ms

Dopamine signal: RPE (reward prediction error) from tracking error

Update frequency: 200 Hz (asynchronous, event-driven)

Memory: 2 KB synaptic weights (on-chip Loihi 2)

Buffering Strategy: LLM Command $\rightarrow$ FIFO Queue (depth: 4 commands) $\rightarrow$

If new command: MPC re-plans with hot-start If buffer empty: MPC continues with last command + spiking correction

5.2 Performance Results

Stability Analysis (Sweep over LLM Latency):

LLM Latency (ms)

No Compensation MPC Only MPC + Spiking 50 ✓ Stable ✓ Stable ✓ Stable 80 ✗ Oscillatory ✓ Stable ✓ Stable

100 ✗ Unstable Δ Marginal ✓ Stable 150 ✗ Unstable ✗ Oscillatory ✓ Stable 200 ✗ Unstable ✗ Unstable Δ Marginal 250 ✗ Unstable ✗ Unstable Δ Marginal Predictive Error vs. Horizon: Horizon (ms) MPC Error (°) MPC + Spiking Error (°) Improvement 50

0.82 ± 0.12

0.51 ± 0.09

38%

100

2.14 ± 0.35

1.23 ± 0.21

43%

150

4.51 ± 0.68

2.04 ± 0.33

55%

200

8.23 ± 1.12

3.51 ± 0.47

57%

Key Finding: The spiking corrector reduces prediction error by up to 57% at 200 ms horizon, enabling stable control up to 200 ms LLM latency.

Full System Experimental Results

6.1 Embedded System Benchmarks

Hardware Setup (Actual Measurements):

Component Device Utilization Power LLM inference Jetson Orin (6x A78AE)

42% CPU, 68% GPU

12.5 W

Primitive cache Jetson Orin (LPDDR5) 6.2 GB Included above

SNN decoder Loihi 2 8% of 1M neurons

0.8 W

MPC + corrector Loihi 2 + ARM core 15% CPU, 12% SNN

1.2 W

Communication PCIe <5% bandwidth

0.1 W

Total (active) Both modules -

14.6 W

Real-Time Performance:

Metric Value Constraint Control loop frequency

500 Hz Fixed by sensor updateEnd-to-end latency (LLM to muscle) 45 ± 8 ms

1,000 trials Jitter (max-min)

12 ms 99th percentile

Deadline misses (>100 ms)

0.3%

Cache miss cases

6.2 Language-Driven Task PerformanceTask 1: Reaching to Visual Target Command: "Grasp the red cube" (position randomized in 30 cm³ workspace)

Result: 92% success (n=100), 1.8 cm final error, 45 ms E2E latency

Task 2: Dynamic Disturbance Rejection Command: "Hold position" Disturbance: 5 N step load at t=500 ms

Result: 35 ms recovery time, 2.3° max deviation

Task 3: Multi-Joint Coordination Command: "Stir a liquid in a circular motion" (wrist radius=5 cm, elbow compensation)

Result: 0.12 rad phase error, jerk=8.4 m²/s³ (vs 15.2 for baseline)

Task 4: Zero-Shot Generalization (Unseen Commands)

Command Type Example Success Rate (n=50)

Synonym variation "Fetch the cube" instead of "grasp" 96%

Speed modifier "Reach very slowly" 94%

Novel motion "Move like a gentle wave" 88%

Combined "Slowly wave to the left" 90%

Average - 92%

6.3 Comparison with Baseline Systems

System E2E Latency Error Power Zero-Shot Hardware CL-MAP (ours) 45 ms 1.8°

14.6 W 92%

Jetson+Loihi

LLM + PID

180 ms 5.2°**35 W 61%**

2× Jetson LLM + MPC

160 ms 3.1°**38 W 68%**

2× Jetson Scripted + SNN N/A (no LLM) 2.1°

6.2 W N/A

Loihi only Human performance ~200 ms <1°

~20 W (brain) 100% Biological

7. Ablation Study (Quantified)

Configuration Error (°)

Max Latency (ms)

Power (W)

Cache Hit Rate Full CL-MAP 1.78 200 14.6 78%

- Predictive compensation 4.20 80 12.8 78%
- Spiking corrector 3.12 120 13.5 78%
- Hybrid (GRU-only)

2.12 200 18.4 78%

- Primitive cache (LLM every step)

1.92 150 22.5 100%

- Fine-tuning (base LLaMA)

2.85 200 14.6 45%

- Quantization (FP16 instead of 4-bit)

1.80 200 28.3 78%

Insights from Ablation: Predictive compensation is non-negotiable for >80 ms latency tolerance Spiking corrector provides 1.34° error reduction at 200 ms latency Hybrid GRU-SNN reduces power by 21% vs GRU-only with same error Primitive cache reduces power by 35% with only 0.14° error penalty Fine-tuning improves cache hit rate from 45% to 78% (critical for real-time)

Statistical Validation

Experimental Design: Trials per condition: n = 100

Commands per trial: 5 different (reach, grasp, push, stir, wave)

Total data points: $100 \times 5 \times 3 \text{ latencies} \times 4 \text{ configurations} = 6,000 \text{ trials}$

Tracking Error Distribution (Full CL-MAP):

Statistic Value Mean 1.78°

Median 1.65°

Standard deviation 0.52°

95% CI [1.68°, 1.88°]

95th percentile 2.81°

Maximum (cache miss)

3.45°

Latency Distribution (LLM → Muscle Contraction):

Statistic Cache Hit Cache Miss Mean 38 ms 72 ms Median

37 ms 68 ms

Std dev

7 ms

12 ms

95th percentile

49 ms 92 ms

Statistical Significance Tests: Comparison t-statistic p-value Significance CL-MAP vs LLM+PID 8.42 <0.001 *** CL-MAP vs LLM+MPC 6.73 <0.001 *** CL-MAP vs Scripted+SNN 2.15

0.034 *

MPC+Spiking vs MPC only 4.98 <0.001 *** Effect Size (Cohen's d):

CL-MAP vs LLM+PID: d = 2.34 (very large)

CL-MAP vs Scripted+SNN: d = 0.68 (medium)

Summary of Contributions

Contribution Result Significance First real-time LLM-to-muscle system

45 ms E2E latency

Enables language-driven physical agents Custom synthetic datasets 40k + 1M samples fully reproducible Embedded implementation
14.6 W total power Deployable on edge hardware Predictive proprioception

200 ms latency tolerance

LLM speed no longer bottleneck Hybrid GRU-SNN 1.8° error, 0.8 μ J/step
efficiency trade-off Zero-shot generalization 92% on unseen commands
Semantic transfer to motor control

Results**Primary Results**

Metric Value End-to-end latency (LLM to muscle) 45 ± 8 ms

Tracking error 1.78°

Zero-shot generalization 92%

Total power consumption

14.6 W

Control loop frequency

500 Hz**Task Performance**

Task Result Reaching ("Grasp the red cube")

92% success, 1.8 cm error

Disturbance rejection (5 N load)

35 ms recovery

Multi-joint coordination

0.12 rad phase error

Unseen commands 92% average success

Comparison with Baselines

System Latency Error Power

CL-MAP

45 ms 1.8°

14.6 W

LLM + PID

180 ms 5.2°

35 W

LLM + MPC

160 ms 3.1°

38 W

Key Findings

Real-time translation achieved: 45 ms end-to-end latency High accuracy: 1.8° tracking error (vs 4.2° for PID baseline)

Robust generalization: 92% success on unseen verbal commands

Energy efficient: 0.8 μ J per step on Loihi 2 Latency tolerance: Stable control up to 200 ms LLM delay with predictive compensation

Final Conclusion: CL-MAP is the first system to demonstrate real-time, direct LLM-to-

joint translation with physical muscle actuation on embedded hardware.

Conclusion

This paper presented CL-MAP, is the first real-time system that directly translates natural language commands from an AI cortex powered by the Large Language Model into physical muscle activation signals for the joint control. The proposed hierarchical architecture, validated on the 7-DOF anthropomorphic limb with pneumatic artificial muscles, achieved

45 ms end-to-end latency, 1.8° tracking error, and 92% zero-shot generalization to the

unseen verbal commands while consuming only of 14.6 W on embedded neuromorphic hardware (Jetson Orin + Loihi 2). The hybrid GRU-SNN and neuromuscular decoder reduced energy consumption by 68% compared to GRU-only and approaches, and the predictive proprioceptive module combining MPC with the cerebellar-like spiking corrector enabled stable control up to 200 ms LLM latency, reducing the prediction error by 57% at 200 ms horizon. All training data were generated in-house and using simulation and inverse dynamics optimization, ensuring a copyright-safe and fully to reproducible approach. CL- MAP demonstrates, for the first time, that a language-driven AI and cortex can directly and robustly command physical muscle systems in the real time, opening pathways toward intelligent, compliant, and linguistically interactive and physical agents such as humanoid robots, intelligent prosthetics, and the assistive devices.

References

- [1] A. Brohan, Y. Chebotar, C. Finn, et al., "Do as I can, not as I say: Grounding language in robotic affordances," in Proceedings of the Conference on Robot Learning (CoRL), 2022.
(SayCan - LLM for robotic planning)
- [2] J. Liang, W. Huang, F. Xia, et al., "Code as policies: Language model programs for embodied control," in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2023. (Code-as-Policies - LLM code generation for robots)
- [3] J. Huebotter, P. Lanillos, M. van Gerven, and S. Thill, "Spiking neural networks for continuous control via end-to-end model-based learning," *Neuromorphic Computing and Engineering*, vol. 6, p. 024004, 2026. (SNNs for multi-DOF robotic arm control)
- [4] M. Davies, A. Wild, G. Orchard, et al., "Advancing neuromorphic computing with Loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021. (Intel Loihi 2 neuromorphic hardware)
- [5] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012, pp. 5026–5033. (Physics simulation environment)
- [6] S. Dey, A. Das, S. Dey, and S. Mukhopadhyay, "PrePARE: Predictive proprioception for agile failure event detection in robotic exploration of extreme terrains," in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2022. (Predictive proprioception framework)
- [7] T. G. Sugar, "McKibben artificial muscle can be in accordance with the Hill skeletal muscle model," in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2002. (Hill-type muscle model for PAMs)
- [8] E. J. Hu, Y. Shen, P. Wallis, et al., "LoRA: Low-rank adaptation of large language models," in International Conference on Learning Representations (ICLR), 2022. (Fine-tuning LLMs efficiently)
- [9] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019. (Training SNNs with surrogate gradients)
- [10] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019. (Inverse dynamics optimization)