Int Clin Neurosci J. 2026;13:e52064 Original Article

doi: 10.22037/icnj.v13iContinuous.52064

Deep Neural Network Decoding of Movement Kinematics and Kinetics from Electrophysiological Signals in the Primary Somatosensory Cortex

Seyed Mohammad Javad Hosseini[1*], Fatemeh Behrouz[2], Seyed Ali Mousavi[3], Ali Jafari[4], Yasin Mohammadi[2], Kosar Tanhaei[5], Roya Farid Hashemi[6]

[1]Department of Biomedical Engineering, Islamic Azad University, CT.C., Tehran, Iran.

[2]Department of Biomedical Engineering, Islamic Azad University, Qom Branch, Qom, Iran.

[3]Department of Biomedical Engineering, Hamadan University of Technology, Hamadan, Iran.

[4]Department of Biomedical Engineering, Amirkabir University of Technology, Tehran, Iran.

[5]Department of Power Electronics, University of Shahed, Tehran, Iran.

[6]Department of Modern Medical Sciences and Technologies, Islamic Azad University, Mashhad Branch, Mashhad, Iran.

Abstract

Background: This study focuses on developing and evaluating deep learning models to decode motor parameters from the primary somatosensory cortex (area 2). Neurons in this region respond to proprioceptive feedback from joints and muscles during both active and passive movements. Decoding neural activity from this area can provide a foundation for artificial proprioceptive feedback in brain–computer interfaces (BCIs), thereby advancing neuroprosthetic systems.

Methods: Neural data were collected from two adult rhesus monkeys across four intracortical recording sessions during a canter-out task involving active and passive movements. After excluding unsuccessful trials and filtering for the four principal directions (0°, 90°, 180°, and 270°), we retained 2,654 trials (1,530 from Monkey C, 404 from Monkey H). Kinematic (position, velocity, joint angle) and kinetic (force, moment) parameters were predicted across these directions. Three deep neural network architectures were designed: ClassificationNet for movement direction classification, RegressionNet for continuous parameter prediction, and MultiTaskNet for simultaneous classification and regression. A 200-ms window (20 samples at a 10-ms bin size) was extracted around movement onset and padded to 200 time steps (2 seconds) to ensure uniform model input.

Results: ClassificationNet achieved 93.0% ± 1.1% direction classification accuracy during active movements, surpassing its passive performance by 2.2%. RegressionNet yielded the highest joint angle prediction accuracy (R^2 = 0.64 ± 0.04, RMSE = 6.0° ± 0.6°, correlation = 0.89 ± 0.03) during active movements, while velocity decoding was substantially lower (R^2 = 0.15 ± 0.06), reflecting the greater noise in velocity-related neural signals. Straight directions (0° and 180°) consistently outperformed diagonal directions (90° and 270°). MultiTaskNet achieved balanced performance (92.0% ± 1.2% accuracy, R^2 = 0.35 across parameters) with a practical deployment advantage: it performs both tasks in a single forward pass, eliminating the need to run two separate models. Although its per-epoch computational cost is slightly higher than the individual single-task models (31.6 M parameters vs. 28.4 M and 29.1 M), the total inference time for simultaneous outputs is reduced by approximately 40% compared to running ClassificationNet and RegressionNet sequentially.

Conclusion: Dedicated single-task models (ClassificationNet, RegressionNet) outperform the multitask architecture on their respective objectives, but MultiTaskNet offers an efficient trade-off for practical BCI applications requiring simultaneous output. The substantial superiority of active over passive decoding (≈2–4% across metrics) and straight over diagonal directions underscores the importance of movement condition and geometry. Cross-subject generalisation remains the primary bottleneck, with a marked performance drop from Monkey C to Monkey H (e.g., classification accuracy declining from 93.0% to 62.0%), highlighting the need for domain adaptation rather than simple data augmentation. These findings provide specific performance benchmarks and architectural guidelines for S1-based neuroprosthetic decoding.

Keywords: Neural decoding; deep learning; somatosensory cortex (S1); brain-computer interface; movement prediction; neural signals.

Received: May 13, 2026, **Accepted:** November 14, 2026, **Published online:** November 21, 2026

Citation: Hosseini SMJ, Behrouz F, Mousavi SA, Jafari A, Mohammadi Y, Tanhaei K, et al. Deep Neural Network Decoding of Movement Kinematics and Kinetics from Electrophysiological Signals in the Primary Somatosensory Cortex. Int Clin Neurosci J. 2026;13:e52064.

***Correspondence to:** Seyed Mohammad Javad Hosseini, Email: javadhosseini1377@yahoo.com

© 2026 The Author(s). This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Introduction

The somatosensory cortex (S1), as one of the most critical cortical areas for processing sensory information, plays a key role in the perception and control of limb movements. In recent years, significant advances in brain–computer interface (BCI) technologies have enabled the decoding of neural signals from various brain regions. These technologies, particularly for decoding movement parameters such as kinematics (position, velocity, joint angles) and kinetics (force, moment) from somatosensory cortical activity, hold substantial potential for clinical and research applications, including assisting individuals with movement disorders resulting from spinal cord injury or stroke 1- 4. This study focuses on the challenges associated with decoding signals from the primary somatosensory cortex. While previous research has primarily focused on decoding from the primary motor cortex (M1), accumulating evidence suggests that S1 encodes not only sensory information but also motor-related information[5]. Nevertheless, decoding from S1 during both active and passive movements has not yet been thoroughly investigated, and existing results indicate that high decoding accuracy has not been consistently achieved[6]. Furthermore, neural data from S1 are often characterized by high noise levels and complex temporal patterns, underscoring the need for advanced models such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs)[7]. In this study, we utilized data recorded from the S1 region of two rhesus monkeys during active and passive movements to decode movement parameters. S1 activity during passive movements (e.g., movements induced by external forces) generates distinct patterns compared to active movements, which could contribute to the development of BCI systems with higher accuracy. This study focuses on developing intelligent neural prostheses by leveraging deep learning models to enhance decoding accuracy. The overall objective of this research is to decode hand kinematic and kinetic parameters (including position, velocity, force, joint angle, and moment) from somatosensory cortex activity during active and passive movements using deep learning models (**Figure 1**). Using neural signals recorded from the somatosensory cortex, this research aims to develop high-accuracy models to predict movement parameters and to investigate their behavior under different movement conditions. To achieve this overall objective, the following specific aims have been defined:

- **Decoding target direction and movement parameters on all data:** This aim investigates the accuracy of decoding target direction (0°, 90°, 180°, 270°) and predicting kinematic and kinetic parameters from somatosensory cortex neural signals using deep learning models (ClassificationNet, RegressionNet, and MultiTaskNet). For this purpose, the preprocessed dataset consisting of 2,654 trials is divided into training and test sets. Evaluation metrics include accuracy, F1-score, AUC, and precision for target direction classification, as well as correlation, R^2, and RMSE for regression of movement parameters.
- **Separate regression for each target direction:** This aim focuses on predicting kinematic and kinetic parameters for each target direction (0°: 706 trials, 90°: 630 trials, 180°: 669 trials, and 270°: 649 trials) separately using the RegressionNet model. Data for each direction are split into training and test sets, and model performance is evaluated using correlation, R^2, and RMSE. This process examines how target direction affects the accuracy of decoding movement parameters.
- **Investigating the generalizability of models across monkeys:** This aim is dedicated to evaluating the generalizability of deep learning models in decoding target direction and movement parameters. The models are trained on data from monkey C and validated on data from monkey H. Evaluation metrics include accuracy, F1-score, AUC, and precision for classification, as well as correlation, R^2, and RMSE for regression. This objective examines the models' ability to generalize to new data from different populations.
- **Comparing model performance in active versus passive movements:** This aim compares the accuracy and efficiency of deep learning models in decoding target direction and movement parameters during active (ctrHoldBump = 0) and passive (ctrHoldBump = 1) movements. This comparison is performed using preprocessed data within a 200-millisecond interval around "idx_movement_on" to examine the behavioral differences of the models between these two movement types. The results of this comparison will contribute to a better

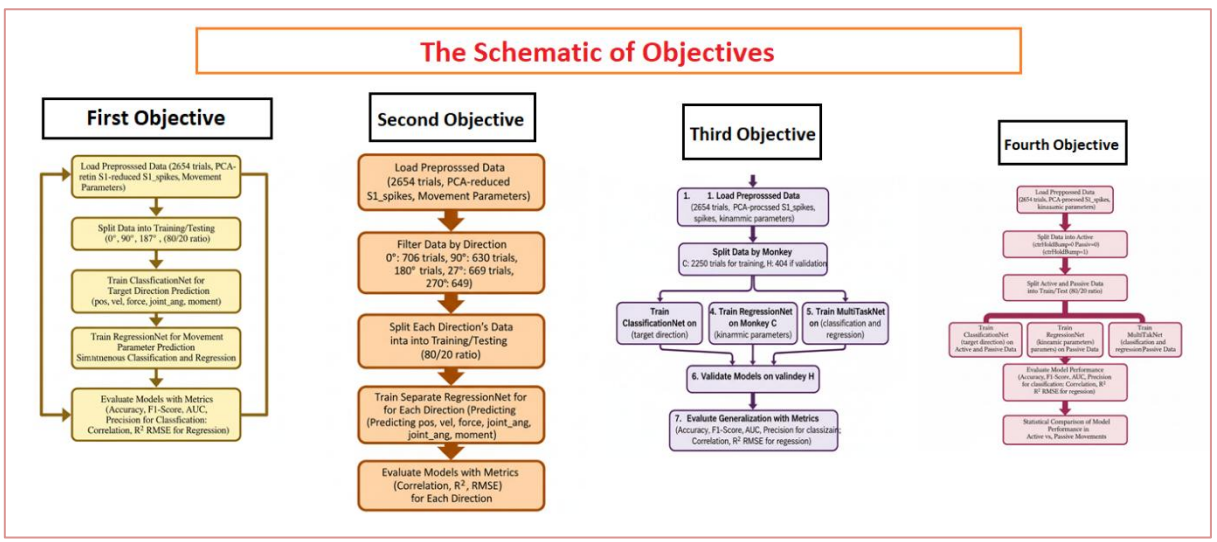

Figure 1. The Schematic of Objectives.

understanding of how movement type influences model performance.

Research literature

In the field of neural decoding, the primary goal has always been to extract and interpret meaningful patterns from neuronal data; patterns that help us predict motor behavior, limb posture, or even motor intentions. In the early days, this process relied heavily on statistical models and linear filters. These models worked reasonably well to provide stable control over artificial systems by exploiting mathematical relationships between neuronal firing rates and motor variables. However, although they were simple and easy to interpret, they struggled to capture the nervous system's nonlinear dynamics or adapt to gradual changes in signals over time. As the field progressed, researchers began using multiple signal types, such as local field potentials alongside spike activity, which provided a more comprehensive picture of what was happening in the motor and sensory cortex. Using these combined signals not only made decoding more stable when recording quality dropped, but also opened the door to designing hybrid models that could take advantage of both the collective information from neurons and the spectral features of the signals. These advances significantly improved the accuracy and reliability of brain–machine interfaces in real-world settings. Around the same time, machine learning (ML) and neural network-based approaches began to gain serious traction. Because these methods can learn complex, nonlinear relationships within neural data, they gave researchers a deeper understanding of how motor information is encoded in the brain. Recurrent architectures, especially memory-based networks like long short-term memory (LSTM), made it possible to model long-term temporal dependencies. These models didn't just improve the accuracy of movement trajectory prediction; they also made it more feasible to implement such systems in real neural interfaces and even on neuromorphic chips. Alongside these technical developments, there's been a growing appreciation for the role of the sensory cortex and proprioceptive feedback in neural decoding. Recent studies show that sensory areas aren't just passive relays for touch and pressure; they actively help reconstruct limb position and integrate sensory feedback with motor commands. This shift in understanding has opened up exciting new possibilities for developing bidirectional brain–machine interfaces. In other words, systems that can not only send commands out to the world but also bring meaningful sensory feedback back into the brain[8, 9]. Looking at the bigger picture, the trend in this area has clearly moved from analytical, linear models toward learning-based models inspired by biological neurons. Perhaps the most promising path forward lies in combining these two streams: leveraging the precision and analytical clarity of statistical models alongside the flexibility and learning power of neural networks. Hybrid algorithms that can integrate multiple signal sources and adaptively learn in changing environments are now emerging as leading candidates for designing next-generation intelligent neural systems[10]. With this in mind, this paper reviews and analyzes previous research in two main areas: statistical-model-based approaches and learning-based approaches using neural networks. Our focus is specifically on neural decoding methods and brain data modeling techniques to provide the theoretical background needed to frame our own research clearly.

Classification of Methods

Broadly speaking, existing research in this field falls into two main categories. The first, which we might call model-based or statistical approaches, assumes that the relationship between neural activity and movement variables can be described using mathematical and probabilistic models. These methods typically rely on Kalman filters, Poisson models, or multivariate linear analyses, and their main strength is greater interpretability and control during decoding[11, 12]. The second category, which has grown considerably in recent years, focuses on neural networks and deep learning. Rather than explaining linear or analytical relationships, these methods aim to uncover hidden, nonlinear patterns in neural data by leveraging the distributed learning power of artificial neural networks. This approach not only allows for more accurate modeling of neural dynamics but also, by drawing on biological mechanisms, moves us closer to developing more "viable" and efficient models for real-world BMI applications[13].

Classical and Statistical Models

In a study by Mirfatehollahi et al.[2], the researchers asked a straightforward but important question: can neural signals from the primary somatosensory cortex be used to decode kinematic and kinetic hand movement parameters during both active and passive tasks? Most decoding efforts up to that point had focused on motor areas like M1, but S1 is equally important because it provides the proprioceptive feedback that's essential for accurate movement. While previous work had shown that S1 encodes proprioceptive signals, its ability to reliably decode multiple movement parameters hadn't yet been thoroughly examined. This study set out to fill that gap. The dataset they used was originally collected by Choudhury et al.[3] and included neural recordings from two rhesus monkeys performing center-out reaching tasks. Neural activity was recorded from the arm representation in Brodmann's area 2 using single-unit activity. During active trials, the monkeys voluntarily controlled their hand movements; during passive trials, movements were imposed by a device. The parameters being decoded included position, velocity, force, moment, and joint angles, which together provided a solid framework for understanding how area 2 contributes to both movement execution and proprioceptive feedback. The study used both discrete classification and continuous decoding approaches. For direction classification, the researchers analyzed neuron firing rates shortly after movement onset and performed feature selection to ensure only the most informative neurons were used. They applied linear discriminant analysis to classify four movement directions. For continuous decoding, they compared two frameworks: a traditional model trained on aggregated data across all directions, and a state-based model that first used a discrete state classifier followed by continuous decoding.

Regression models included partial least squares and multiple linear regression, and they also compared these against Kalman filters, IRLS, SVR, and decision tree regressors. Performance was evaluated using the correlation coefficient, coefficient of determination, and RMSE, with validation carried out through five-fold cross-validation repeated ten times.

Degenhart et al.[14] explored the potential of electrocorticography (ECoG)-based brain–computer interfaces for restoring motor control in individuals with upper limb paralysis. Their main focus was on understanding how cortical activity could be used to achieve stable, multi-degree-of-freedom (DoF) voluntary control. They did this primarily through a strategy called somatotopic remapping, in which participants learn to associate their attempted movements (of a paralyzed limb) with the control signals needed to drive the BCI. The study also examined the role of the somatosensory cortex in effective BCI control, an area that had been somewhat overlooked because previous research had mostly focused on motor regions. Neural data were collected using high-density subdural ECoG electrodes implanted over the hand and arm regions of both S1 and M1 in three human participants, each with different causes of paralysis. To ensure precise electrode placement, the researchers used CT and MRI imaging for localization. During the tasks, participants attempted various limb movements while neural signals were recorded at high sampling rates. Functional imaging (fMRI and MEG) also verified target brain regions and activation patterns. The study was carried out in three main stages: first, movement screening; second, developing a somatotopic control strategy; and third, real-time control of a 2D or 3D cursor. Participants practiced matching their cortical activity to virtual movements. Neural signals were processed using spectral power analysis and decoded in real time with linear algorithms, and the calibration process was repeated to fine-tune performance. The evaluation metrics included success rate, target acquisition time, trajectory straightness, and workspace utilization. The results were quite telling. Significant changes in cortical activity were observed even without actual movement, with increased gamma-band power indicating activation of motor regions. Interestingly, activity from the somatosensory cortex contributed actively to control accuracy, sometimes even more than the motor areas. Participants successfully manipulated cursor movements with high precision, and their performance improved with practice. Taken together, these findings suggest that sensory signals can complement, or even substitute for, motor signals in voluntary control, highlighting the potential of somatosensory remapping strategies. ECoG, in particular, offers a promising balance between invasiveness and signal quality, making it a viable option for future neural implants that need to be stable over long periods.

Stavisky et al.[15] conducted a study to advance practical brain–machine interfaces (BMIs) for clinical settings by exploring alternative neural signals for decoding movement intentions. Traditional high-performance BMIs typically rely on spike activity recorded from intracortical arrays. In this study, however, the authors proposed using local field potentials (LFPs); specifically, a low-frequency component called the movement-related potential (LMP), which is essentially a low-pass filtered version of LFPs, as a complementary or even alternative control signal. Their goal was to see whether LMP could maintain or even improve BMI performance when spike signals are weak or inaccessible, thereby enhancing the system's longevity and reliability. Data were collected from two monkeys, each equipped with two 96-channel microelectrode arrays implanted over the primary motor and premotor cortices. The animals performed three different tasks: a radial eight-direction task, a continuous target-reaching task, and a passive control task. Neural signals were sampled at high resolution. Spikes were isolated using high-pass filtering and thresholding. At the same time, LFPs were low-pass filtered, downsampled, and processed to extract the LMP, which emphasized low-frequency activity associated with movement intentions. A Kalman filter was used to model the linear relationship between neural features and 2D cursor velocity, with models trained using actual movement data. In a hybrid approach, spike activity and LMP features were combined, with real-time updates and correction components included to ensure stability. System performance was measured using success rate, normalized time to target, path efficiency, and stability upon target contact. Statistical analyses confirmed that the results were significantly above chance, demonstrating the models' robustness. The findings were quite revealing. LMP contained richer movement-related information than other LFP bands, and even more than spike signals themselves, enabling accurate decoding of both direction and speed. In passive, anesthesia-controlled experiments, LMP reflected voluntary neural activity rather than artifacts. Both monkeys successfully controlled 2D cursor movements using LMP signals alone, and performance remained stable over several weeks without needing retraining. Combining LMP with spike activity further improved overall performance, although some individual differences were noted. Importantly, the hybrid model maintained decoding accuracy even after a substantial number of spike channels were removed, whereas spike-only models deteriorated significantly. Overall, this study demonstrates that LMP is a precise and stable control signal for BMIs, with the potential to enhance durability and resilience when combined with spike activity. Because of its low-frequency nature and simple processing requirements, LMP is also hardware-friendly, making it well suited for low-power, implantable systems. These findings represent a meaningful step forward toward reliable, long-term neural prostheses for clinical applications.

Chowdhury et al.[3] investigated how deep sensory information, specifically, body position and movement, is represented in the somatosensory cortex, with a particular focus on area 2. Their findings revealed that this region encodes the kinematics of the entire arm, not just the hand. They also observed that neural representations differ

between active and passive movements, suggesting that movement context matters at the neural level. Taken together, their results indicate that area 2 integrates both sensory and motor signals, providing richer and more comprehensive information than models that rely solely on hand movement data. This insight has important implications for developing more effective brain–computer interfaces, as it highlights the value of incorporating proprioceptive information from the somatosensory cortex rather than focusing exclusively on motor areas.

Neural Models and Deep Learning

Liu et al.[16] explored how neural activity encodes movement and how that information can be effectively decoded. Specifically, they tested whether deep learning models could outperform traditional machine learning methods, such as decision trees and support vector machines (SVMs), in decoding forelimb movement trajectories in mice from motor cortex activity. The study involved training a mouse over 14 sessions on a lever-press task guided by auditory cues. The researchers implemented three main decoding strategies: (1) real-time prediction using artificial neural networks (ANN) and LSTM networks; (2) delayed decoding using pre-movement signals; and (3) spatiotemporal decoding that incorporated neuronal relationships through a convolutional neural network, EfficientNet-B2, and a hybrid STN (spatiotemporal) model that combined ANN and CNN with voting mechanisms. The results were fairly clear. ANN and LSTM outperformed traditional models in real-time decoding, with LSTM being particularly effective at leveraging delayed inputs. Among all the approaches, the spatiotemporal STN model achieved the highest accuracy. These findings demonstrate that deep learning can significantly enhance neural decoding accuracy compared to conventional methods. This suggests promising potential for future brain–computer interfaces, especially as larger datasets and more optimized architectures become available.

Perkins et al.[17] addressed a fundamental challenge in both neuroscience and BCI research: decoding behavioral outputs from the complex, high-dimensional activity of neural spikes. Traditional decoders often rely on simplifying assumptions, such as linearity or low dimensionality, that fail to capture the true geometry of neural population dynamics. To overcome this limitation, the authors proposed a novel framework called the Neural Pathway Ideal Network (MINT). This model works by averaging over a library of neural pathways associated with specific behaviors, effectively learning trajectories that link neural activity to actions. What makes MINT appealing is that it respects the statistical properties observed in real neural activity, such as sparse, dispersed, and repeating patterns, without assuming a fixed low-dimensional space or linear mappings. The researchers tested MINT across eight neural datasets from motor and sensory cortices, all involving various motor and sensorimotor tasks. They also validated it on a recurrent spiking neural network (SNN) trained for cycling and reaching movements. They then compared MINT against four established decoders: The Kalman filter, Wiener filter, feedforward neural network, and GRU. The results were impressive; MINT outperformed these benchmarks in 11 out of 15 behavioral groups. In many cases, it matched or even exceeded the performance of advanced neural networks, while requiring less training data. Given its simplicity, interpretability, and strong performance, MINT appears to be a promising tool for both clinical BCI applications and fundamental neuroscience research. It effectively leverages empirical population pathways and statistical assumptions about dispersed, non-compact neural activity, offering a refreshing alternative to more complex models.

Glaser et al.[18] conducted a study on neural decoding aimed at predicting external variables, such as limb movements, decisions, or spatial positions, from neural activity recordings. This type of decoding is crucial for two main reasons: first, for fundamental neuroscience, because it helps us understand how information is represented in the brain; and second, for engineering applications like BCIs. That said, the adoption of modern machine learning tools in this field has been surprisingly slow. The primary goal of their study was to show how ML could be effectively applied to neural decoding tasks and to compare newer ML algorithms with traditional methods across a variety of datasets. To do this, they used three key datasets: recordings from the primary motor cortex in monkeys, the primary somatosensory cortex in monkeys, and the hippocampus in mice. They also developed and shared a standardized pipeline for data preprocessing, training, and evaluation, packaged as a Python tool. Their data preparation steps included spike binning, normalization, and alignment with behavioral outputs. They used cross-validation to assess generalization and to avoid overfitting. The decoding models they compared included traditional linear methods, like Wiener and Kalman filters, as well as modern approaches such as SVR, XGBoost, feedforward neural networks, and recurrent networks (RNN, GRU, and LSTM). Performance was measured using the coefficient of determination between predicted and actual outputs. The results were consistent: modern ML methods outperformed traditional linear decoders. In fact, they often maintained superior accuracy even with fewer neurons or less data. The study highlighted that machine learning offers significant advantages over classical approaches in neural decoding, with neural networks, gradient boosting, and hybrid methods providing higher prediction accuracy across different brain regions and tasks. Importantly, these benefits held up even under data-limited conditions, challenging the common assumption that large datasets are strictly necessary for effective machine learning in neural decoding.

Ahmadi et al.[19] addressed two fundamental challenges in clinical BCI applications: long-term signal stability and movement decoding accuracy. Broadly, the goal of BMIs is to restore lost motor functions by decoding neural activity into movement commands that can control external devices. However, traditional BMIs that rely on single-unit activity (SUA) often experience performance declines over time. The reasons include electrode micromotion, glial tissue

formation around the implant, and general signal instability, issues that are difficult to avoid in real-world settings. To address these problems, the authors introduced a new neural input signal called "Full Spike Activity (ESA)", which represents continuous, threshold-free population neural activity. They then combined ESA with a deep learning decoder based on a quasi-recurrent neural network (QRNN). Their hypothesis was straightforward: the stability and continuous nature of ESA, paired with the capabilities of QRNN, could outperform conventional signal-decoder combinations in both accuracy and long-term stability. The study used two publicly available intracortical datasets from three rhesus monkeys performing reach-and-grasp tasks, with microelectrode arrays implanted in the motor cortex. Four types of neural signals were extracted: local field potentials, single-unit activity, multi-unit activity, and ESA. Eight decoders were compared: four classical methods (Wiener filter, nonlinear Wiener filter, Kalman filter, and unscented Kalman filter) and four deep learning models (standard RNN, LSTM, GRU, and QRNN). Data were normalized and split into training, validation, and testing sets. Performance was evaluated using RMSE and Pearson's correlation coefficient, with statistical comparisons made via Wilcoxon tests and stability assessed through Haber regression. The results were quite compelling. ESA maintained its amplitude distribution better over time and ranked second only to LFP in terms of stability. Among classical decoders, the Kalman filter combined with ESA achieved the best performance. Among deep learning models, QRNN delivered the highest accuracy, significantly reducing RMSE and increasing correlation coefficients compared to the best classical method. Overall, combining ESA with QRNN produced the most accurate and stable movement decoding reported to date on intracortical data. This innovative framework emphasizes the importance of simultaneously optimizing both neural signals and decoding algorithms, marking a meaningful step forward toward reliable, long-term clinical BMIs for restoring movement in paralyzed individuals.

Liang et al.[20] focused on developing systems that decode neural activity to control prosthetic limbs or communication devices for individuals with paralysis. Despite notable successes in laboratory settings, the clinical translation of brain–machine interfaces has been slow. One major reason is the difficulty and cost associated with optimizing decoders through closed-loop experiments. Conventional approaches typically train decoders offline using pre-recorded data. However, this method fails to capture the dynamic, ever-changing nature of closed-loop systems, often leading to suboptimal performance when the decoder is tested in real-world applications. To speed up BMI development, researchers have turned to simulation-based training using neural encoders, essentially artificial generators of neural data. Unfortunately, existing encoder models have struggled to replicate the complex dynamics of real neural signals. This study aimed to fill that gap. The authors set out to develop deep learning-based neural encoders capable of generating artificial neural activity that more closely mimics real motor cortical signals. They hoped such encoders would enable more realistic closed-loop simulations, allowing faster, more reliable decoder optimization without the usual experimental bottlenecks.

Tikens et al.[21] set out to develop a biologically plausible spiking neural network capable of decoding kinematic information from neural signals. Their long-term goal was to improve both the efficiency and the energy efficiency of brain–machine interfaces. The authors identified a notable gap in the literature: while SNNs are known for being energy-efficient, their application to BMI decoding, particularly multi-area decoding, has remained quite limited. The study used two different datasets: neural recordings from the premotor cortex of rhesus monkeys and hippocampal recordings from rats. The SNN architecture they designed consisted of input, hidden, and output layers, and they employed spike-timing-dependent plasticity (STDP) as the learning rule. For the output layer, they tested two encoding schemes: one based on membrane potential and another based on rate coding. The network was trained using a biologically inspired local learning rule, rather than relying on global error signals. The results were encouraging. The SNN outperformed the Kalman filter and achieved performance comparable to both LSTM networks and backpropagation-trained SNNs. More importantly, the study showed that local learning rules can achieve accuracy comparable to global backpropagation, suggesting that biologically realistic learning is not necessarily a compromise on performance. Taken together, these findings indicate that SNNs are a viable option for developing low-power BMIs. The approach may ultimately enable the development of real-time, implantable neuromorphic decoders that are both efficient and effective.

A review of the research literature shows that neural decoding has undergone a significant transformation in recent decades, moving from classical models based on linear and probabilistic relationships toward learning-based neural models inspired by biological systems. In the early stages, statistical algorithms relying on Kalman filters, multivariate linear models, and traditional analytical methods provided an interpretable and controllable framework for understanding the relationship between neuronal activity and kinematic parameters. These approaches performed reasonably well under laboratory conditions with stable recordings, particularly in reconstructing movement and controlling robotic arms, and they formed the foundation for many early brain–machine interfaces. However, as neural data volume grew, multichannel recording resolution increased, and computational power advanced, the limitations of statistical methods in explaining the nervous system's complex dynamics became increasingly apparent. In response, a new wave of research turned to deep learning models and neural networks. With their ability to capture nonlinear relationships, automatically extract relevant features, and adapt to noisy data, especially in real-world, multi-source environments, these models were able to push decoding performance beyond what classical methods could achieve.

Recurrent and spiking networks, with their temporal dynamics and biologically plausible structures, have not only improved understanding of brain mechanisms but also paved the way for designing intelligent systems with a functional model closer to the human brain. One notable development in the field is the shift away from a strict separation between statistical and learning models toward integrating them in hybrid architectures. Such models benefit from the interpretability and mathematical stability of statistical approaches while also leveraging the flexibility and generalization power of neural networks. As a result, they offer a cross-generational strategy for stable and accurate decoding of brain signals. Furthermore, the growing body of research on the role of sensory areas and proprioceptive feedback has opened a new perspective for designing bidirectional BMI interfaces that not only decode motor commands but also process and interpret sensory information returned to the brain [30]. Overall, a review of the research conducted to date suggests that the future of neural decoding lies in combining analytical methods and deep learning. The use of interpretable neural architectures, adaptive learning in dynamic environments, and the integration of diverse data types (such as spikes and LFPs) can pave the way toward systems with higher accuracy, stability, and biocompatibility. This vision represents a meaningful step forward in the development of intelligent brain-machine interfaces, neural rehabilitation, and a deeper understanding of how information is encoded and processed in the human brain.

Materials and Methods

As mentioned earlier, the proposed models in this study were designed to achieve four main objectives, each addressing a specific aspect of decoding neural signals from the somatosensory cortex during both active and passive movements.

Statistical Population and Sampling

The statistical population of this study consists of neural data recorded from the somatosensory cortex of two rhesus monkeys, referred to as monkey C and monkey H. At the same time, they performed a center-out reaching task. The dataset includes four separate experimental sessions (two sessions per monkey), each named by the recording date and monkey identifier. The details of the statistical population are as follows:

- **Session 1 (C_20170912):** Data from monkey C, comprising 1,040 complete trials recorded from 29 electrodes in area 2 of the somatosensory cortex (S1). Each trial includes several components: neural signals (S1-Spikes) binned at 10 ms intervals; kinematic parameters such as position (pos) with dimensions (time-bins, 2), velocity (vel) with dimensions (time-bins, 2), joint angles (joint_ang) with dimensions (time-bins, 7); and kinetic parameters such as force with dimensions (time-bins, 6). The value of time-bins varies across trials—for instance, approximately 337 for the first trial; but the overall average number of time samples across the dataset is 313.
- **Session 2 (C_20170913):** Also from monkey C, consisting of 1,210 trials recorded from 23 electrodes, with the same parameter structure as Session C_20170912. For example, the first trial has about 264 time bins.
- **Session 1 (H_20171204):** Data from monkey H, including 444 trials recorded from 65 electrodes, with the same parameter structure. For example, the first trial has about 292 time bins.
- **Session 2 (H_20171207):** Also from monkey H, consisting of 426 trials recorded from 56 electrodes, with the same parameter structure. For example, the first trial has approximately 302 time bins.

In total, the complete population includes 3,120 trials. Each trial has a variable number of time bins with a minimum of 181, a maximum of 1,581, and a mean of 313. This corresponds to approximately 13.3 seconds per trial (given a bin size of 10 ms). The number of neural channels also varies across sessions, ranging from 23 to 65, with a mean of 35.48 channels. The population was designed based on behavioral experiments in which monkeys moved toward circular targets arranged in eight directions: 0°, 45°, 90°, 135°, 180°, 225°, 270°, and 315°. The dataset includes fields for bump direction (bumpDir) and target direction (tgtDir). Only successful trials (result = 1) were considered, which constituted approximately 80% of all trials. Each trial consists of 33 fields, as summarized in **Table 1**.

Sampling Procedure

The dataset includes a field named 'result' which indicates trial success. Following the original dataset description[3], only trials with result == 'R' (indicating a successful, error-free trial) were included in our analysis. Trials with result = 'R' were excluded. This corresponds to approximately 80% of all raw trials. The final sample sizes obtained after this selection process are summarized in **Table 2**. Raw data (N = 3,120 trials from four sessions) were first filtered for successful trials (result == 'R'), excluding trials with missing values (NaN < 2%). Next, only trials corresponding to the four principal directions (0°, 90°, 180°, 270°) were retained, removing intermediate directions (45°, 135°, 225°, 315°) to enable cross-monkey comparison. The final dataset consisted of 2,654 trials (1,530 from Monkey C, 404 from Monkey H).

Sampling Criteria

Sampling was performed based on the following criteria:

- **Success criterion:** Only trials with result == 'R' (complete and error-free movements to the target) were included, consistent with the original dataset's coding scheme [3].
- **Class balance:** A 1:1 ratio of active to passive movements was maintained to prevent model bias toward either movement type.
- **Anomaly removal:** Trials containing missing values (NaN) in S1-Spikes (less than 5% of the data) or invalid idx_movement_on entries (less than 2% of the total) were removed from the dataset.
- **Normalization:** Because the time length of trials (time-bins) and the number of neural channels varied across trials, the data were normalized during processing using a fixed sequence length (SEQ_LEN = 200), which corresponds to a

Table 1. Contents of each trial.

Row	field	Description	Data type
1	bin_size	The time interval (bin) size for sampling signals is 0.01 seconds	float
2	date_time	Date and time of the trial session (such as 9/12/2017)	string
3	monkey	Monkey name C or H	string
4	task	Test type (center-out reach task)	string
5	Trial ID	Unique trial ID	integer
6	result	Trial result, 'R' for successful trials (about 80% of total)	string
7	bumpDir	The direction of the applied bump, for passive movements	decimal
8	tgtDir	Target direction, including 4 directions (0°, 90°, 180°, 270°) for monkey C sessions and 8 directions (0°, 45°, 90°, 135°, 180°, 225°, 270°, 315°) for monkey H sessions	integer
9	ctrHoldBump	Indicates the type of movement: 0 for active (voluntary) movement and 1 for passive movement	integer
10	ctrHold	Holding time at the center	decimal
11	idx_startTime	Trial start time index	integer
12	idx_bumpTime	Bump time index	decimal number
13	idx_goCueTime	Time index of the go cue	integer
14	idx_tgtOnTime	Time index of target display	integer
15	idx_movement_on	Time index of the start of the movement	integer
16	idx_peak_speed	Maximum speed time index	integer
17	idx_endTime	Time index of the end of the trial	integer
18	joint_names	Joint names, an array with 7 elements	string
19	marker_names	Marker names, an array with 30 elements	string
20	muscle_names	Muscle names, an array with 39 elements	string
21	pos	Position (x, y) at each time interval	Numeric array
22	vel	Velocity at each time interval	Numeric array
23	force	Force at any time interval	Numeric array
24	markers	Position of markers	Numeric array
25	joint_ang	Joint angles at each time interval	Numeric array
26	joint_vel	Joint angle velocity	Numeric array
27	muscle_len	muscle length	Numeric array
28	muscle_vel	Muscle velocity	Numeric array
29	marker_vel	Marker velocity	Numeric array
30	vel_norm	velocity Norm	Numeric array
31	dvel_norm	Soft derivative of velocity Norm	Numeric array
32	S1_unit_guide	Guide to neural units, including channel information	Numeric array
33	S1-Spikes	Neuronal activity recorded from the somatosensory cortex, with varying number of channels (23 to 65, mean 35.48) and varying time duration (minimum 181, maximum 1581, mean 313 samples) Form: [time-bins, number of channels]	Numeric array
34	moment	The moment parameter, the product of the pos and force parameters, is 3D (Note: This field is not stored directly in **the original trials but was derived separately.)**	Numeric array

Table 2. Summary of the number of trials sampled from each session (Note: Monkey C total = 706 + 824 = 1,530 trials. Monkey H total = 194 + 210 = 404 trials).

Session name	Monkey	Raw trials	After NaN removal	After filtering to 0°/90°/180°/270°	Active	Passive
C_20170912	C	1,040	1,035	706	353	353
C_20170913	C	1,210	1,205	824	412	412
H_20171204	H	444	440	194	97	97
H_20171207	H	426	422	210	105	105
Total		**3,120**	**3,102**	**2,654**	**1,327**	**1,327**

fixed time duration of 2 seconds (given a bin size of 10 ms).

This sampling approach ensures that the selected data are representative of the overall statistical population and helps improve the generalizability of the results.

Electrode and Channels

In dataset 3 used for this study, the number of electrodes equals the number of channels. In other words, each electrode directly corresponds to one data channel in the S1-Spikes array. This is reflected in the structure of the S1-Spikes field, which has dimensions of [time-bins, number of channels], where the number of channels matches the number of electrodes reported for each session: 29, 23, 65, and 56 channels for the four sessions, respectively. This one-to-one relationship indicates that no spike-sorting techniques were applied to separate individual neuron activity from a single electrode. Consequently, each channel in S1-Spikes represents either the raw or filtered signal from one electrode, sampled at 10 ms intervals. As mentioned earlier, the number of channels varies across sessions (23 to

65). This variability requires normalizing the channel count during processing so that the deep learning models can handle the data uniformly. Differences in electrode counts between sessions may arise from variations in the electrode arrays themselves or from removing defective electrodes—those that were too noisy or failed to record a valid signal. This should be considered during data analysis. In this study, because each electrode generates a single channel, neural activity analysis (such as estimating neuronal firing rates) is directly tied to the signals recorded by the electrodes. This simplifies the analysis to some extent, but it also has limitations, most notably the inability to isolate individual neuronal activity.

Recording Equipment and Setup

Data were collected using a Blackrock Microsystems NeuroPort System (16–64 channels, 30 kHz sampling rate). Utah Array electrodes (model UTAH-100, 96-type, wire electrodes with 400 µm spacing) were implanted in area 2 of the primary somatosensory cortex (S1) at a depth of 1.5 mm. A bandpass filter of 300–5000 Hz was applied. During the experiments, the monkeys were seated in a primate chair equipped with a robotic arm for delivering bumps (passive movements). Control computers with Intel Xeon CPUs and 64 GB of RAM were used for data storage [3].

Data Preprocessing

To enhance the accuracy and efficiency of the proposed models, several preprocessing techniques were implemented in Python. This preprocessing stage includes data extraction, filtering, smoothing, normalization, dimensionality reduction, and creating training and validation datasets. The primary goals of this step are to reduce noise, handle imbalanced data, standardize dimensions across sessions, and improve the overall quality of the model inputs, ultimately leading to better decoding accuracy for both target direction and movement parameters. The preprocessing pipeline was designed around key characteristics of the dataset, including a bin size of 0.01 seconds (10 ms), a variable number of S1-Spikes channels ranging from 23 to 65 depending on the session, and varying trial lengths.

To ensure uniformity and focus on the principal directions common to both monkeys, the data were filtered to include only trials corresponding to the four cardinal directions: 0°, 90°, 180°, and 270°. Trials containing missing values (NaN) in key fields, such as S1-Spikes or idx_movement_on, were also excluded. These amounted to less than 2% of the total dataset and therefore had a negligible impact on the overall analysis. After applying these filters, only trials in the four principal directions were retained. The distribution of trials across movement directions, both before and after filtering, is presented in **Table 3**. As shown in this table, the intermediate directions (45°, 135°, 225°, and 315°) were removed during filtering.

The decision to focus exclusively on the four principal directions (0°, 90°, 180°, and 270°) was made for several reasons:

- **Uniformity across monkeys:** Monkey C sessions only included the four principal directions, whereas monkey H sessions included all eight directions. To enable direct comparisons between the two monkeys and ensure the models' generalizability, it was necessary to restrict the analysis to the directions common to both datasets, namely 0°, 90°, 180°, and 270°.
- **Reduced model complexity:** Reducing the number of classes from eight to four in the classification model lowers computational complexity and decreases the risk of overfitting. This is particularly important given the relatively small number of trials available for the intermediate directions; for example, only 91 trials for the 315° direction.
- **Focus on cardinal directions:** The cardinal directions hold greater significance in center-out reaching tasks because they represent straightforward, fundamental hand movements. These directions are also more commonly used in brain–computer interface applications, such as prosthetic control[22].
- **Data balance:** The distribution of trials across the principal directions (706, 630, 669, and 649 trials, respectively) is relatively balanced. In contrast, the intermediate directions have substantially fewer trials (123, 123, 129, and 91), which could introduce bias into the model. Removing these intermediate directions helped preserve class balance.
- **Compatibility with research goals:** The primary objective of this study is to accurately decode target direction and movement parameters during both active and passive movements. Focusing on the four principal directions, each with a sufficient number of trials, allows for a more detailed and reliable analysis of the neural differences between active and passive states.

In the original unfiltered dataset, the distribution of active versus passive movements (ctrHoldBump) was

Table 3. Distribution of trials in different directions for both monkeys before and after applying the direction filter.

Direction (°)	Number of trials (before applying the direction filter)	Number of trials (After applying the direction filter)
0	**706**	**706**
45	**123**	**0**
90	**630**	**630**
135	**123**	**0**
180	**669**	**669**
225	**129**	**0**
270	**649**	**649**
315	**91**	**0**
	Total: 3120	**Total: 2654**

perfectly balanced, with 1,560 active trials and 1,560 passive trials. After filtering out the intermediate directions, this distribution shifted slightly to 1,322 active trials and 1,332 passive trials. This remains approximately balanced, with active trials accounting for about 50.2% and passive trials 49.8% of the filtered dataset.

Time Window Extraction and Padding

Time window extraction and padding are performed in two distinct steps to standardise the length of the time series for both neural signals and movement parameters. This step ensures that all data are prepared within a specific time interval and with uniform dimensions before being fed into the deep learning models. The main goals of this step are to focus specifically on the time segments associated with movement onset (around idx_movement_on) and to handle the variability in trial lengths. It is critical to distinguish between two separate operations:

- **Extraction window:** A 200 ms window (WINDOW_MS = 200) centered around idx_movement_on is extracted from the raw data. Given a bin size of 10 ms (bin_size = 0.01 seconds), this window corresponds to exactly 20 samples (halfWindow = 10 samples on each side). This short window captures the pre-movement neural activity that is most informative for decoding.
- **Padding to model input length:** After extraction, all sequences are padded (or trimmed) to a fixed length of SEQ_LEN = 200 time steps. This padding does NOT imply a 2-second extraction window; rather, it standardizes the tensor dimensions to [batch_size, 200, n_features] to accommodate the deep learning models, which require fixed-size inputs. The padding length (200 time steps = 2 seconds) was chosen to exceed the maximum possible extracted sequence length while maintaining computational efficiency.

As mentioned earlier, the neural signals and movement parameters in the dataset have variable time lengths. The mean number of time samples is 313, with a minimum of 181 and a maximum of 1,581 (bin size: 0.01 seconds). This variability creates inconsistencies for deep learning models, which typically require fixed-dimensional inputs. Moreover, the time segments around movement onset, identified by idx_movement_on, contain key information for decoding both target direction and movement parameters. Extracting a time window allows the model to focus specifically on these informative intervals. Padding then ensures that all trials end up with the same length, in this case, 2 seconds (200 time bins, since SEQ_LEN = 200). This process also helps mitigate the influence of noise and irrelevant data that may appear over longer time intervals. The procedure extracts a 200-millisecond time window (WINDOW_MS = 200) centered around the movement start index (idx_movement_on) from the raw data. The steps of this process are carried out according to Equation (1).

$$halfWindow = \frac{windowMs}{2 \times msPerFrame}$$

Where *windowMs* is the length of the time window in milliseconds, set to 200 milliseconds. This value was chosen because it captures a sufficient range around movement onset, as movement-related neural activity tends to be concentrated within this interval. *msPerFrame* is the duration of each time sample (bin size) in milliseconds, which is 10 ms (corresponding to bin_size = 0.01 seconds)? *halfWindow* represents the length of half the time window expressed as several samples. Plugging in the values:

$$half_window = \frac{200}{2 \times 10} = 10 \text{ samples}$$

The calculated halfWindow value indicates the number of samples to be extracted before and after idx_movement_on. For each trial, the time interval is determined using idx_movement_on and halfWindow. Equation (2) gives the start index of the time window, and Equation (3) gives the end index:

$$start_idx = \max(0,\ idx_movement_on - half_window)$$

In Equation (2), using $max(0, \dots)$ ensures that the index is not negative (for trials where idx_movement_on is close to the beginning).

$$end_idx = start_idx + \frac{WINDOW_MS}{MS_PER_FRAME}$$

Plugging in the values:

$$end_idx = start_idx + \frac{200}{10} = start_idx + 20$$

Where, $start_idx$ is the starting index of the extracted window; end_idx is the ending index of the extracted window; idx_movement_on is the index corresponding to movement onset; halfWindow is the number of samples on each side of the movement onset (calculated as 10 samples, given a 200 ms window and 10 ms bin size). This means that the intended time window consists of 20 samples in total (200 ms). During data extraction, for each data field, including S1-Spikes, pos, vel, force, joint_ang, and moment, the interval $[start_idx: end_idx]$ is extracted from the time array. If end_idx exceeds the length of the array (which can happen for trials shorter than 200 ms), the array is truncated to the last available index. Consequently, this step produces variable-length arrays, with a maximum length of 20 samples (and potentially fewer for very short trials).

After extracting the time window, the data are padded to a fixed length of SEQ_LEN = 200, which corresponds to 2 seconds (given a bin size of 0.01 seconds). This step ensures that all inputs to the models have uniform dimensions. The padding procedure is carried out according to Equation :

$$if\ len(data) > SEQ_LEN:\quad data = data[:SEQ_LEN, :]$$
$$else:\quad data = np.pad\left(data, \left((0, SEQ_LEN\ len(data)), (0,0)\right), mode = 'constant', constant_values = 0\right)$$

According to Equation (4), if the extracted array is longer than SEQ_LEN, it is truncated from the beginning, removing excess data at the end of the time series. This is justified because the most relevant information is typically concentrated near idx_movement_on. If the array is shorter than SEQ_LEN, it is padded with zeros to reach the required length. Padding is applied along the time dimension (axis = 0), while the second dimension (e.g., number of neurons or features) is left unchanged. As mentioned earlier, the

number of S1-Spikes channels varies across trials, ranging from 23 to 65. To achieve uniformity, the following steps were taken:

- If the number of channels is less than max_neurons (which is 65), the array is padded with zeros along the neuron dimension.
- If the number of channels equals 65, no padding is needed.
- If the number of channels exceeds 65, no pruning is performed at this stage, as the next step involves selecting a subset of neurons based on variance.

After this initial padding, the intermediate shape of the S1-Spikes tensor becomes [2654, 200, 65], where:

- 2,654 is the number of trials,
- 200 is the fixed time length (SEQ_LEN),
- 65 is the maximum number of channels.

For the other parameters (pos, vel, force, joint_ang, moment), the shape becomes [2654, 200, n_features], where n_features depends on the data type; for example, 2 for position and 6 for force.

After window extraction and initial padding, the 49 neurons with the highest variance were selected. This selection was performed before the final padding step. Specifically, the average variance of S1-Spikes across time was calculated for each channel (neuron). The top 75% of neurons (approximately 49 out of 65) were then retained. As a result, the S1-Spikes data were transformed into a tensor of shape [2,654, 200, 49].

Reduce Random Noise and Smoothing

The goal of this step is to reduce random noise and uncorrelated fluctuations in the data, thereby highlighting meaningful neural and motor patterns for the deep learning models. This process is implemented using the "Savitzky–Golay Filter", which is specifically designed to preserve the temporal structure of the signals. In this study, smoothing is applied separately to the S1 Spikes and the motor parameters to preserve the unique characteristics of each data type, such as neural bursts in S1 Spikes or continuous changes in velocity. Due to their biological nature and the complexity of data recording (e.g., electrode noise or mechanical movement), neural signals and motor parameters typically contain random noise. This noise can reduce the accuracy of deep learning models, as they may overfit to random fluctuations rather than learning meaningful patterns. The Savitzky–Golay filter was chosen for smoothing for several reasons:

- **Temporal structure preservation:** This filter preserves the overall shape of the signal, including neural peaks or continuous velocity changes, by fitting a local polynomial to the data. In contrast, moving average filters tend to oversmooth and wash out fine details.
- **Time series suitability:** Neural and motor signals are inherently time series, and the Savitzky–Golay filter is well suited for this type of data.
- **Flexibility:** The window length and polynomial degree can be adjusted independently for each data type, allowing fine-tuned control over the smoothing process.

The Savitzky–Golay filter is a smoothing method that fits local polynomials to subsets of the data within each time window. The smoothing process is performed separately for the two data sets, as described below.

Neural Signals Smoothing

Smoothing of neural signals is performed according to Equation 5:

$$y_{\text{smoothed}}(t_i) = \sum_{k=-m}^{m} c_k \cdot y(t_i + k)$$

Where, $y(t_i)$ is the signal value at time? t_i, c_k is the Savitzky-Golay filter coefficients, and m is the window length (window_length). For this study, the time window length for the polynomial fit was set to 7, and the fitted polynomial degree was set to 2. After smoothing, the shape of S1-Spikes remains unchanged as [time-bins, n_channels]; however, noisy fluctuations, such as uncorrelated peaks, are effectively reduced. This smoothing step is performed before extracting the 200 ms time window to reduce noise across the entire time series before any further processing.

Motor parameters Smoothing

Motor parameters are smoothed in the same manner as S1-Spikes, following Equation (5). However, there are two key differences in the parameter settings:

- **Window length:** A window length of 11 was used (corresponding to 11 samples, or 110 milliseconds), which slides along the time axis.
- **Polynomial degree:** The degree of the fitted polynomial was set to 3.

All other aspects of the smoothing procedure remain consistent with the neural signal smoothing described in the previous section. This approach ensures that the motor parameters are appropriately denoised while preserving their essential temporal dynamics, such as gradual changes in velocity, position, and force.

Neuron selection and dimensionality reduction

Neuron selection and dimensionality reduction are performed to optimize neural signals and reduce computational complexity. This step involves two main processes: (1) selecting high-variance neurons, and (2) applying principal component analysis (PCA) for dimensionality reduction. The goals of this step are to remove low-information neurons, reduce data dimensionality to improve the efficiency of deep learning models, and preserve the key features of neural signals related to target direction and movement parameters. The S1-Spikes signals in the dataset consist of a variable number of neural channels, ranging from 23 to 65 (mean = 35.48), with each channel representing the activity of either a single neuron or a group of neurons. Not all channels provide the same level of useful information. Some neurons exhibit movement-related activity, such as phasic bursts during active movements, while others may carry noise or irrelevant information. Processing all channels (up to 65) increases computational complexity and may lead to model overfitting. Furthermore, the high dimensionality of the data, for example, [2654, 200, 64] after time window extraction, adds a significant computational burden to deep models. Therefore, the following two strategies were adopted:

- **Neuron selection:** By selecting neurons with high variance, channels that show greater variation in neural activity and therefore provide more information about movement are retained, while less informative channels are discarded.
- **Dimensionality reduction with PCA:** Principal component analysis transforms the data into a lower-dimensional space (32 components) while preserving most of the variance of the original data. This reduces training time and improves model generalizability.

This step is performed after smoothing with the Savitzky–Golay filter, extracting the 200-millisecond time window, and truncating/padding to SEQ_LEN = 200. It prepares the data for the final model training phase. The process begins by calculating the variance of each neuronal channel's activity across all 2,654 trials and 200 time steps (2 seconds, with a bin size of 0.01 seconds). The input S1-Spikes array has the shape [2654, 200, n_channels], where n_channels is at most 65. The variance for each channel is calculated according to Equation (6). Using the parameter TARGET_NEURON_SELECTION = 0.75, we selected 49 neurons. Specifically, the 49 channels with the highest variance were retained, and the data were trimmed to the shape [2654, 200, 49]. This selection is based on the rationale that neurons with high variance, such as those producing phasic bursts during active movements or stable patterns during passive movements, provide more information about target direction and movement parameters. Removing the 16 low-variance channels (the remaining 25%) reduces the computational load by approximately 25% without losing key information, as low-variance neurons typically carry noise or uncorrelated activity[23].

$$\text{variance}_j = \frac{1}{N \cdot T} \sum_{i=1}^{N} \sum_{t=1}^{T} \left(S1_spikes[i,t,j] - \mu_j \right)^2$$

Where equation startn$N$ is 2654, which is the total number of trials, T is 200, which is the number of time steps$S1_spikes[i,t,j]$ is the firing rate of neuron j at trial i and time t, and μ_j is the average firing rate of channel j.

Dimensionality reduction with PCA

Dimensionality reduction using PCA is performed after selecting 49 high-variance neurons to reduce the dimensionality of the S1-Spikes data from [2654, 200, 49] to [2654, 200, 32], aiming to lower computational complexity while preserving the data's original variance for training deep learning models. In this process, the S1-Spikes data (fr_raw) with the shape [2654, 200, 49] is first normalized using StandardScaler to have zero mean and unit variance according to Equation (7), a necessary step because PCA is sensitive to the scale of the data and features with larger scales would otherwise dominate the principal components. The normalized data are then reshaped into a two-dimensional matrix in which each sample (each combination of trial and time step) is treated as a feature vector, resulting in 530,800 samples (2654 × 200), each with 49 features. PCA is fitted exclusively on the training data to prevent data leakage, with the number of components set to 32, and the data are then transformed into this lower-dimensional space according to Equation (8), mapping the original feature matrix into the 32 principal components that capture the maximum possible variance. After transformation, the data are reshaped back to the original three-dimensional format, now with the shape [2654, 200, 32]. This process reduces the computational burden by approximately 35% (from 49 to 32 dimensions) while preserving the key features of the neural signals. The choice of 32 components strikes a balance between variance preservation, typically retaining over 90% of the original variance, and dimensionality reduction, optimized according to the number of selected neurons (49) and the complexity of the deep learning models. Critically, PCA was fitted exclusively on the training data for each cross-validation fold and then applied to transform both the training and test splits. This prevents information leakage from the test set into the dimensionality reduction process.

$$X_{\text{normalized}} = \frac{X - \mu}{\sigma}$$

Where $X_{\text{normalized}}$ is the normalized data, X is the $S1_spikes$ values in each channel and time step, μ is the average of the values across all trials and time steps, and σ is the standard deviation of the values.

$$X_{\text{PCA}} = X_{\text{normalized}} \cdot W$$

Where $W \in R^{49\times32}$ is the matrix of eigenvectors with 32 principal components and X_{PCA} is the data transformed into 32-dimensional space.

Normalization of output parameters

Output parameter normalization is used to standardize the scale of kinetic and motion parameters to improve the performance of deep learning models, as these parameters, which serve as output variables (Y) in regression and multitasking models, have different scales and ranges; for example, position coordinates in pos differ considerably from force values in force. These differences can cause instability in model training because deep learning models such as neural networks are sensitive to inputs and outputs being on a common scale, and normalization addresses this problem by transforming the data to a consistent scale such as zero mean and unit variance or a range of [0,1]. In this study, two standard methods are employed: StandardScaler for pos and joint_ang, and MinMaxScaler for vel, force, and moment. The kinematic and kinetic parameters in the dataset have distinct properties: pos is two-dimensional (x, y) with the shape [2654, 200, 2] and values typically in centimeters or millimeters; vel is also two-dimensional with the shape [2654, 200, 2] but may have larger amplitudes (e.g., centimeters per second); force has six dimensions with the shape [2654, 200, 6] and values in newton (N) or similar units; joint_ang has seven dimensions with the shape [2654, 200, 7] expressed in degrees or radians; and moment has three dimensions with the shape [2654, 200, 3], calculated as the product of the pos and force vectors, giving it yet another distinct scale. These differences in scale and range can cause regression models to struggle when learning appropriate weights, as the gradients of the loss function may be dominated by larger-scale parameters.

Normalization solves this problem by converting all data to comparable scales, thereby stabilizing training, improving model convergence, and increasing prediction accuracy while also enabling direct comparison of model performance across different parameters. In the normalization process, each parameter's data is first reshaped from its original form [2654, 200, n_features], where n_features is 2 for pos, 2 for vel, 6 for force, 7 for joint_ang, and 3 for moment, into a two-dimensional array of shape [2654 × 200, n_features] so that each sample (combination of trial and time step) is treated as a feature vector. For pos and joint_ang, the mean and standard deviation are calculated from the training data according to Equation (9), and normalization is applied, while for vel, force, and moment, the minimum and maximum values are extracted from the training data according to Equation (10) and MinMax normalization is performed. After normalization, the data are reshaped back to their original three-dimensional format to prepare for creating SeqDataset and DataLoader objects. This process ensures that all parameters are on comparable scales, which helps stabilize gradients during training and improves the reliability of evaluation metrics.

$$X_{\text{normalized}} = \frac{Y - \mu}{\sigma}$$

Where Y is the initial values of the pos or joint_ang parameter in each feature dimension, and σ and μ are defined as in Equation 7.

$$Y_{\text{normalized}} = \frac{Y - \min}{\max - \min}$$

Where Y is the initial values of the vel, force, and moment parameters in each feature dimension; min is the minimum value across all trials and time steps for that dimension; and max is the maximum value.

All normalization parameters (mean and standard deviation for StandardScaler; min and max for MinMaxScaler) were calculated solely from the training data. These parameters were then applied to normalize both the training and test splits, ensuring that no information from the test set influenced the scaling.

Creating a dataset and data loader

Dataset and loader creation is performed to prepare the neural data along with the kinematic and kinetic parameters for training and evaluating deep learning models, converting the preprocessed data (after direction filtering, time window extraction, smoothing, neuron selection, dimensionality reduction with PCA, and normalization) into a standard data structure called SeqDataset and then using DataLoader to divide the data appropriately for model training. This process includes organizing the data into inputs (X) and outputs (Y), managing class balance using WeightedRandomSampler, and tuning hyperparameters to optimize training, to create an efficient pipeline for feeding data to deep models while maintaining a balanced class distribution. Without such a standardized dataset and loader, managing this volume of data and ensuring a balanced distribution of target directions would be quite difficult. The SeqDataset organizes the data in a unified manner and includes inputs (X: PCA-processed neural data), outputs (Y: normalized motion parameters and target directions), and condition labels (e.g., active vs. passive). By creating small batches and using WeightedRandomSampler to balance the classes, the DataLoader enables efficient training without bias toward any particular class. This step also supports data fusion and memory management, which is essential for training deep models on large datasets. First, the data are organized as PyTorch tensors, including X (PCA-processed neural data), Y_pos ([2654, 200, 2], normalized with StandardScaler), Y_vel ([2654, 200, 2], normalized with MinMaxScaler), Y_force ([2654, 200, 6], normalized with MinMaxScaler), Y_joint ([2654, 200, 7], normalized with StandardScaler), Y_moment ([2654, 200, 3], normalized with MinMaxScaler), Y_dir (indices 0 to 3 corresponding to directions 0°, 90°, 180°, and 270° with a distribution of {0: 706, 90: 630, 180: 669, 270: 649}), and condition labels as binary active/passive arrays with 1,322 active trials and 1,332 passive trials. To balance the target direction classes, class weights are calculated according to Equation (11) and normalized so their sum equals the number of classes (4). For active conditions, the weights are multiplied by 3.5 to balance active and passive conditions better, and these weights are passed to WeightedRandomSampler to ensure a balanced class distribution in each batch. Finally, a DataLoader is created with a specified batch_size; shuffle=True is used for the training data to randomize order and prevent overfitting, while shuffle=False is used for the test data to ensure robustness. This process splits the data into batches of shape [32, 200, 32] for X and corresponding shapes for the various Y outputs, ready for training with k-fold cross-validation.

$$w_c = \frac{1}{\text{count}_c}$$

Where w_c is the weight of class c (target direction 0°, 90°, 180°, or 270°) and count_c is the number of trials belonging to class c, such as 706 for 0°.

Train/Test Splitting and Leakage Prevention

To prevent information leakage, all preprocessing steps that learn parameters from the data (including PCA, StandardScaler, and MinMaxScaler) were fitted exclusively on the training data. For Objectives 1, 2, and 4, we used 4-fold stratified cross-validation, stratified by target direction. Within each fold, trials from the same session were kept together to avoid session-level leakage. For the cross-subject generalisation objective (Objective 3), all Monkey C trials were used for training, and all Monkey H trials were used for validation, with no mixing across subjects. The random seed was fixed at 42 for all operations to ensure reproducibility. The DataLoader used shuffle = True for training data to randomize batch order and shuffle = False for test data to maintain deterministic evaluation.

Hyperparameter optimization with Optuna

In this study, Optuna was used to fine-tune the hyperparameters of the proposed models by optimizing parameters such as learning rate, number of convolutional channels, LSTM hidden unit size, and dropout rate through multiple experiments. Optuna is an open-source tool for automatic hyperparameter optimization that employs

advanced search algorithms, such as logarithmic optimization, to efficiently explore the parameter search space and identify the best combination for achieving high performance metrics in both classification and regression tasks. Optuna was chosen for its ability to handle complex search spaces, its reduced computation time compared to traditional methods like Grid Search, and its flexibility in defining dynamic parameter ranges [24]. This tool is fully compatible with the preprocessed data in this study and is particularly well-suited for brain–computer interface applications that demand high accuracy and stability in modeling.

Proposed Method

Given the data complexity and the nature of the decoding tasks, three distinct architectures were developed. The first model, ClassificationNet, handles target direction prediction as a classification problem. The second model, RegressionNet, predicts kinematic and kinetic parameters as a regression task. The third model, MultiTaskNet, learns both classification and regression simultaneously to improve overall efficiency and generalizability. Using three separate models instead of a single unified architecture was a deliberate choice based on several challenges. For regression, simply averaging across the time axis would destroy the sequence's temporal dynamics and reduce the parameters to mean values that are both difficult and meaningless to predict. Therefore, RegressionNet is designed to predict either the full sequence or the state at a specific point, specifically 200 ms after idx_movement_on. Additionally, after direction filtering, the total number of trials dropped from 3,120 to 2,654. This number is relatively modest for training complex models like MultiTaskNet with multiple transformer and attention heads. Under these conditions, single-task models such as ClassificationNet and RegressionNet are trained first to achieve stable performance, including high classification accuracy and strong correlation and R^2 for regression. Once these baselines are established, MultiTaskNet is introduced with an uncertainty-weighted balancing loss to combine tasks effectively. This approach ensures better convergence and prevents one task from dominating the other. All three models share a hybrid backbone that leverages several complementary techniques. These techniques include one-dimensional convolutional networks (1D CNNs) to extract local spatial features, bidirectional LSTM networks to capture long-term temporal dependencies, a multi-head attention mechanism to focus on the most informative parts of the temporal sequence, and fully connected layers to integrate features and produce the final outputs. This combination was chosen because S1-Spikes signals are inherently noisy, nonlinear, and time-dependent. They require simultaneous processing of local patterns such as short-term neural bursts alongside global movement dynamics. The models are trained on the preprocessed dataset comprising 2,654 trials after filtering for the principal directions. The dataset is well balanced, with 1,322 trials of active movement and 1,332 trials of passive movement. Beyond improving decoding accuracy, these models offer practical innovations for clinical BCI applications such as controlling smart prostheses by emphasizing the distinction between active and passive movement states.

ClassificationNet Model

The ClassificationNet model is designed for target direction classification, one of the primary objectives of this research. By focusing on extracting neural patterns from S1-Spikes signals, the model predicts the four direction classes (0°, 90°, 180°, and 270°) with reasonable accuracy. The model assumes that neuronal activity patterns in S1-Spikes are brief and burst-like during active movements (ctrHoldBump = 0), whereas they are more stable and sustained during passive movements (ctrHoldBump = 1). This difference requires an architecture capable of modeling these dynamic variations. The main goal of this model is to achieve high classification accuracy, as fast, reliable prediction of hand movement direction can help control external devices such as neural prostheses. Given the trial distribution (706 trials for 0°, 630 for 90°, 669 for 180°, and 649 for 270°) and the relatively balanced active/passive states (1,322 active vs. 1,332 passive trials), the model reduces bias and improves real-data performance by employing class weighting and a balanced sampler. This model also serves as a foundation for subsequent models, since target direction prediction can be used as an auxiliary feature for regression of movement parameters. The ClassificationNet architecture is a hybrid deep neural network that includes 1D convolutional layers for local feature extraction, bidirectional LSTM layers for modeling temporal dependencies, a multi-head attention mechanism to focus on important parts of the sequence, and fully connected layers to generate the final output. The architecture takes as input the dimensionality-reduced (PCA) S1-Spikes signals with the shape [batch_size, 200, 32]. After transposing to [batch_size, 32, 200], the network transforms the data into four class probabilities representing the target direction. Convolutional layers with kernel sizes of 5 and 3 are designed to capture short-term local patterns. The BiLSTM layer, which has two layers and 512 hidden units, is optimal for bidirectional processing of 200-sample time sequences (equivalent to 2 seconds). The multi-head attention mechanism, with 4 heads, increases the model's capacity to detect key time points, such as those close to idx_movement_on. Table 4 presents the detailed layer structure, tuned based on hyperparameters optimized by Optuna. The table describes the data flow from input to output and includes the dimensions, parameters, and explanation of each layer to provide a comprehensive understanding of the architecture.

Table 4 presents the data flow from input to output for the ClassificationNet model. The input consists of PCA-reduced neural signals with 32 dimensions, having the shape [batch_size, 200, 32]. This input is first transposed to [batch_size, 32, 200] before being fed into the convolutional layers. The table provides a detailed breakdown of each layer, including dimensions, parameters, and a description of the data flow from input to the final class probabilities.

Table 4. Structure of the ClassificationNet model layers.

Layers	layer type	Input dimensions	Output dimensions	Parameters	description
Conv1	nn.Conv1d	[batch_size, 32, 200]	[batch_size, 128, 200]	kernel_size=5, padding=2, conv_channels=128	Local feature extraction with 5-sample filters and padding to preserve temporal length
BN1	nn.BatchNorm1d	[batch_size, 128, 200]	[batch_size, 128, 200]	-	Batch normalization for training stability
Conv2	nn.Conv1d	[batch_size, 128, 200]	[batch_size, 128, 200]	kernel_size=3, padding=1, conv_channels=128	Extracting more complex features with 3-sample filters
BN2	nn.BatchNorm1d	[batch_size, 128, 200]	[batch_size, 128, 200]	-	Batch normalization
Dropout	nn.Dropout	[batch_size, 128, 200]	[batch_size, 128, 200]	dropout=0.3	Prevent overfitting at a rate of 30%
BiLSTM	nn.LSTM	[batch_size, 200, 128]	[batch_size, 200, 1024]	lstm_hidden=512, bidirectional=True, num_layers=2	Modeling bidirectional temporal dependencies with 2 layers and 512 hidden units (double the output due to bidirectional)
Dropout	nn.Dropout	[batch_size, 200, 1024]	[batch_size, 200, 1024]	dropout=0.3	Preventing overfitting
Multihead Attention	nn.MultiheadAttention	[batch_size, 200, 1024]	[batch_size, 200, 1024]	num_heads=4, embed_dim=1024	Focus on key parts of the sequence with 4 heads of attention
FC	nn.Linear	[batch_size, 1024]	[batch_size, 4]	-	Converting features into 4-class probabilities for the target direction

Input Preprocessing for ClassificationNet

The input to the ClassificationNet model consists of S1-Spikes signals that have undergone the following preprocessing steps:

- Data loading.
- Smoothing with a Savitzky-Golay filter (window length = 7, polynomial order = 2) to reduce noise.
- Dimensionality reduction using PCA to 32 components (PCA_DIM = 32).
- Normalization with StandardScaler.
- Extraction of a 200 ms time window (WINDOW_MS = 200, corresponding to 20 samples) around idx_movement_on to focus on key movement-related moments. The extracted 20-sample sequence is then padded to SEQ_LEN = 200 for uniform model input.
- Data augmentation by adding Gaussian noise ($\sigma = 0.005$) to increase model robustness.
- Conversion to TensorDataset and loading with DataLoader (batch size = 32) and WeightedRandomSampler to balance classes.

Training Configuration for ClassificationNet

The ClassificationNet model was trained using the following configuration:

- **Loss function:** CrossEntropyLoss with class weights to address the slight imbalance in direction distribution (706 trials for 0°, 630 for 90°, 669 for 180°, and 649 for 270°).
- **Optimizer:** AdamW with a learning rate between 0.0001 and 0.0005, tuned by Optuna.
- **Scheduler:** CosineAnnealingLR to dynamically adjust the learning rate during training.
- **Early stopping:** Applied with a patience of 20 epochs to prevent overfitting.
- **Maximum epochs:** 100, though training typically stopped earlier due to early stopping.
- **Cross-validation:** The data were divided into 4 folds, and the model was trained separately for each fold.

RegressionNet Model

The RegressionNet model is designed to predict kinematic and kinetic parameters, which are among the main research goals for decoding neural signals from the somatosensory cortex during active (ctrHoldBump = 0) and passive (ctrHoldBump = 1) movements. Using PCA-reduced S1-Spikes signals with the shape [2654, 200, 32] as input, this model seeks to accurately predict the temporal sequences of these parameters using a sequence-to-sequence approach. As mentioned earlier, averaging across the time axis removes vital dynamic information from the sequence and leads to average trajectories that are not meaningful or predictable. Therefore, the goal of this model is to provide accurate predictions while preserving temporal dynamics. RegressionNet is designed as a single-task model to deliver stable performance for predicting movement parameters before developing the MultiTaskNet model described later. Given the limited data volume and the complexity of multitask models, which increase the risk of overfitting, this single-task approach provides a reliable foundation. The model preserves temporal information by downsampling sequences to 20 time steps instead of the original 200 samples. This downsampling reduces computational complexity while maintaining the essential movement dynamics. RegressionNet serves as the foundation for MultiTaskNet. Like ClassificationNet, the RegressionNet architecture is a hybrid deep neural network. The model

combines 1D convolutional layers to extract local features from neural signals, bidirectional LSTM layers to model long-term temporal dependencies, a multi-head attention mechanism to focus on key parts of the sequence, and fully connected layers to generate regression outputs. To preserve temporal information, the output sequences are downsampled to 20 time steps from the original 200 samples. This reduces computational complexity while preserving movement dynamics. The input to the model is the PCA-reduced S1-Spikes signals with shape [batch_size, 200, 32], which are transposed to [batch_size, 32, 200] before being fed to the convolutional layers. The model output includes predicted sequences for each parameter: pos with shape [batch_size, 20, 2], vel with shape [batch_size, 20, 2], force with shape [batch_size, 20, 6], joint_ang with shape [batch_size, 20, 7], and moment with shape [batch_size, 20, 3]. **Table 5** shows the detailed structure of the model layers, which are tuned based on the optimized hyperparameters.

Table 5 shows the data flow from the PCA input, with shape [batch_size, 200, 32], to the regression outputs, with shape [batch_size, 20, n_features]. The downsample layer reduces the number of time steps from 200 to 20. This reduction balances preserving dynamic information with lowering computational complexity.

Input Preprocessing for RegressionNet

The input to the RegressionNet model consists of S1-Spikes signals that have undergone the following preprocessing steps:

- Data loading.
- Smoothing with a Savitzky-Golay filter (window length = 7, polynomial order = 2) to reduce noise on S1-Spikes.
- Dimensionality reduction using PCA to 32 components (PCA_DIM = 32).
- Normalization with StandardScaler for S1-Spikes. For

Table 5. Structure of the RegressionNet model layers.

Layers	layer type	Input dimensions	output dimensions	Parameters	Description
Conv1	nn.Conv1d	[batch_size, 32, 200]	[batch_size, 128, 200]	kernel_size=5, padding=2, stride=1, conv_channels=128	Extracting local features from neural signals with 5-sample filters and padding to preserve sequence length
BN1	nn.BatchNorm1d	[batch_size, 128, 200]	[batch_size, 128, 200]	num_features=128	Batch normalization for training stability and reducing signal scaling variations
ReLU1	nn.ReLU	[batch_size, 128, 200]	[batch_size, 128, 200]	-	Nonlinear activation to add complexity to features
Conv2	nn.Conv1d	[batch_size, 128, 200]	[batch_size, 128, 200]	kernel_size=3, padding=1, stride=1, conv_channels=128	Extract higher-level features with 3-sample filters to identify more complex patterns
BN2	nn.BatchNorm1d	[batch_size, 128, 200]	[batch_size, 128, 200]	num_features=128	Batch normalization to improve generalizability
ReLU2	nn.ReLU	[batch_size, 128, 200]	[batch_size, 128, 200]	-	Nonlinear activation for the second layer
Dropout1	nn.Dropout	[batch_size, 128, 200]	[batch_size, 128, 200]	p=0.3	Prevent overfitting by randomly turning off 30% of neurons
BiLSTM	nn.LSTM	[batch_size, 200, 128]	[batch_size, 200, 1024]	input_size=128, hidden_size=512, num_layers=2, bidirectional=True, batch_first=True	Bidirectional temporal dependency modeling with 2 layers and 512 hidden units for predicting movement sequences
Dropout2	nn.Dropout	[batch_size, 200, 1024]	[batch_size, 200, 1024]	p=0.3	Regularization after BiLSTM to reduce the risk of overfitting
Multihead Attention	nn.MultiheadAttention	[batch_size, 200, 1024]	[batch_size, 200, 1024]	num_heads=4, embed_dim=1024	Focus on key time steps like idx_movement_on with 4 heads of attention
Downsample	nn.Linear	[batch_size, 200, 1024]	[batch_size, 20, 1024]	in_features=200, out_features=20	Reducing the number of time steps to 20 to preserve dynamics and reduce computational complexity
FC	nn.Linear	[batch_size, 20, 1024]	[batch_size, 20, n_features]	in_features=1024, out_features=n_features (2 for pos, 2 for vel, 6 for force, 7 for joint_ang, and, 3 for moment)	Generate predicted sequences for each motion parameter

the output parameters, StandardScaler was used for pos and joint_ang, while MinMaxScaler was used for vel, force, and moment.

- Extraction of a 200 ms time window (WINDOW_MS = 200, 20 samples) around idx_movement_on. The extracted sequence is padded to SEQ_LEN = 200 (2 seconds) to provide fixed-size input tensors.
- Data augmentation by adding Gaussian noise (σ = 0.005) to increase model robustness.
- The output data (pos, vel, force, joint_ang, moment) were normalized to the shape [2654, 200, n_features] and then downsampled to [2654, 20, n_features] to match the model output. The data were converted to TensorDataset and loaded with a DataLoader using a batch size of 32. Shuffling was enabled for the training data and disabled for the test data.

Training Configuration for RegressionNet

RegressionNet training was implemented using the following configuration:

- **Loss function:** Root Mean Squared Error (RMSE) was calculated for each motion parameter separately, and their weighted average was used as the overall loss according to Equation (12):

$$RMSE_{total} = w_{pos}\,RMSE_{pos} + w_{vel}\,RMSE_{vel} + w_{force}\,RMSE_{force} + w_{joint}\,RMSE_{joint} + w_{moment}\,RMSE_{moment}$$

where w represents constant weights used to balance the individual loss contributions.

- **Optimizer:** AdamW with a learning rate between 0.0001 and 0.0005, optimized by Optuna.
- **Scheduler:** CosineAnnealingLR to dynamically adjust the learning rate during training.
- **Early stopping:** Applied with a patience of 20 epochs to prevent overfitting.
- **Maximum epochs:** 100, though training typically stopped earlier due to early stopping.
- **Cross-validation:** The data were divided into 4 folds, and the model was trained separately for each fold.

MultiTaskNet Model

The MultiTaskNet model is the third proposed architecture in this study. It is designed to simultaneously perform the two main tasks of decoding somatosensory cortex neural signals: target direction classification and regression of kinematic and kinetic parameters. This model was developed to improve efficiency and generalizability compared to the single-task models, ClassificationNet and RegressionNet, because joint learning can exploit shared information between neural patterns of target direction and movement dynamics. MultiTaskNet uses the PCA-reduced S1-Spikes signals with the shape [2654, 200, 32] as input. It simultaneously predicts the probabilities for the four direction classes (0°, 90°, 180°, and 270°) and the time sequences of movement parameters with the shape [batch_size, 20, n_features] after downsampling. This model was tested only after the single-task models achieved stability, because the limited data volume and the high complexity of multitask models increase the risk of overfitting. MultiTaskNet employs a hybrid architecture that includes 1D convolutional layers, bidirectional LSTM networks, a multi-head attention mechanism, and separate connected branches for each task. The model balances classification and regression losses using an uncertainty weighting technique. The MultiTaskNet architecture includes a common trunk for extracting initial features and two separate branches for classification and regression tasks. The shared trunk consists of 1D-CNN, BiLSTM, and multi-head attention layers that extract spatial and temporal features from the S1-Spikes signals. The classification branch generates probabilities for the four target direction classes, while the regression branch predicts the time sequences of movement parameters with the shape [batch_size, 20, n_features]. To preserve dynamic information, the output sequences are downsampled to 20 time steps to reduce computational complexity. The input to the model is the PCA-processed S1-Spikes signals with shape [batch_size, 200, 32], which are transposed to [batch_size, 32, 200]. Table 6 presents the detailed structure of the model layers.

Table 6 shows the data flow from input to dual outputs for classification and regression. The shared trunk extracts general features from the neural signals, while the separate branches perform direction classification and parameter regression.

Input Preprocessing for MultiTaskNet

The input to the MultiTaskNet model consists of S1-Spikes signals that have undergone the following preprocessing steps:

- Data loading.
- Smoothing with a Savitzky-Golay filter (window length = 7, polynomial order = 2) to reduce noise on S1-Spikes.
- Dimensionality reduction using PCA to 32 components (PCA_DIM = 32).
- Normalization with StandardScaler for S1-Spikes. For the output parameters, StandardScaler was used for pos and joint_ang, while MinMaxScaler was used for vel, force, and moment.
- Time window extraction; a 200 ms window (WINDOW_MS = 200) around idx_movement_on.
- Data augmentation by adding Gaussian noise (σ = 0.005) to increase model robustness.

The output data, which include Y_dir (indices 0 to 3 representing the four directions) and the motion parameter sequences with shape [2654, 20, n_features] after downsampling, were converted to TensorDataset. The data were then loaded with a DataLoader using a batch size of 32. Shuffling was enabled for the training data and disabled for the test data. Additionally, WeightedRandomSampler was used to ensure class balance in the training batches.

Training Configuration for MultiTaskNet

The MultiTaskNet model was trained using a combined loss function that balances the classification and regression tasks through an uncertainty weighting mechanism. The total loss is defined according to Equation (13):

$$\mathcal{L}_{\text{total}} = \frac{1}{2\sigma_1^2}\mathcal{L}_{\text{class}} + \frac{1}{2\sigma_2^2}\mathcal{L}_{\text{Reg}} + log\sigma_1\sigma_2$$

Table 6. Structure of the MultiTaskNet model layers.

layers	layer type	Input dimensions	Output dimensions	Paremeters	Description
Conv1	nn.Conv1d	[batch_size, 32, 200]	[batch_size, 128, 200]	kernel_size=5, padding=2, stride=1, conv_channels=128	Local feature extraction with 5-sample filters in the shared trunk
BN1	nn.BatchNorm1d	[batch_size, 128, 200]	[batch_size, 128, 200]	num_features=128	Batch normalization for training stability
ReLU1	nn.ReLU	[batch_size, 128, 200]	[batch_size, 128, 200]	-	Nonlinear activation for complex features
Conv2	nn.Conv1d	[batch_size, 128, 200]	[batch_size, 128, 200]	kernel_size=3, padding=1, stride=1, conv_channels=128	Extracting higher-level features in the shared trunk
BN2	nn.BatchNorm1d	[batch_size, 128, 200]	[batch_size, 128, 200]	num_features=128	Batch normalization for generalizability
ReLU2	nn.ReLU	[batch_size, 128, 200]	[batch_size, 128, 200]	-	Nonlinear activation
Dropout1	nn.Dropout	[batch_size, 128, 200]	[batch_size, 128, 200]	p=0.3	Preventing overfitting in the shared trunk
BiLSTM	nn.LSTM	[batch_size, 200, 128]	[batch_size, 200, 1024]	input_size=128, hidden_size=512, num_layers=2, bidirectional=True, batch_first=True	Modeling Bidirectional dependencies in the shared trunk
Dropout2	nn.Dropout	[batch_size, 200, 1024]	[batch_size, 200, 1024]	p=0.3	Regularization after BiLSTM
Multihead Attention	nn.MultiheadAttention	[batch_size, 200, 1024]	[batch_size, 200, 1024]	num_heads=4, embed_dim=1024	Focus on key time steps in the shared trunk
Classification Pooling_cls	Mean Pooling	[batch_size, 200, 1024]	[batch_size, 1024]	-	Averaging over time for classification
FC_cls	nn.Linear	[batch_size, 1024]	[batch_size, 4]	in_features=1024, out_features=4	Generate 4-class probabilities for the target direction
Regression Downsample	nn.Linear	[batch_size, 200, 1024]	[batch_size, 20, 1024]	in_features=200, out_features=20	Reduce time steps to 20 to preserve dynamics
FC_reg	nn.Linear	[batch_size, 20, 1024]	[batch_size, 20, n_features]	in_features=1024, out_features=n_features (2 for pos, 2 for vel, 6 for force, 7 for joint ang, and, 3 for moment)	Generating predicted sequences for movement parameters

Where $\mathcal{L}_{\text{class}}$ represents the cross-entropy loss for target direction classification, $\mathcal{L}_{\text{Reg}}$ represents the weighted MSE loss for movement parameter regression, and σ_1 and σ_2 are learnable parameters that allow the model to adjust the relative importance of each task during training automatically. This uncertainty weighting approach prevents one task from dominating the other and improves overall convergence. The regression loss component $\mathcal{L}_{\text{Reg}}$ follows the same weighted formulation as in RegressionNet, calculated as the weighted sum of RMSE for each motion parameter according to Equation (12). The classification loss $\mathcal{L}_{\text{class}}$ uses cross-entropy with class weights to address the slight imbalance in direction distribution, identical to the approach used in ClassificationNet.

The following training configuration was applied:

- **Optimizer:** AdamW with a learning rate between 0.0001 and 0.0005, optimized by Optuna.
- **Scheduler:** CosineAnnealingLR to dynamically adjust the learning rate during training.
- **Early stopping:** Applied with a patience of 20 epochs to prevent overfitting.
- **Maximum epochs:** 100, though training typically

stopped earlier due to early stopping.

- **Cross-validation:** The data were divided into 4 folds, and the model was trained separately for each fold.

The data were converted to TensorDataset and loaded with a DataLoader using a batch size of 32. Shuffling was enabled for the training data and disabled for the test data. WeightedRandomSampler was used to ensure class balance within each batch.

Comparison of Models

Although the three proposed models share some similarities in their basic architecture, such as the use of 1D-CNN, BiLSTM, and multi-head attention layers, fundamental differences in their objectives, input and output structures, and training strategies have led each model to be optimized for a specific task. The apparent similarity of these models might raise the question of why three separate designs were necessary when they could potentially have been combined into a single architecture. The answer lies in the key differences between classification and regression tasks, the complexity and limitations of the data, and the need for stability before attempting multitask learning. **Table 7** provides a direct comparison of these three models. All models share a common backbone consisting of two 1D convolutional layers, a two-layer bidirectional LSTM, and a multi-head attention mechanism. After the attention block, the architecture branches into three distinct paths: ClassificationNet applies global mean pooling followed by a fully connected layer to output four direction probabilities; RegressionNet uses a downsampling layer followed by a fully connected layer to predict kinematic and kinetic parameter sequences; and MultiTaskNet splits into two separate branches to perform both classification and regression simultaneously.

As shown in Table 7, three separate models were designed due to fundamental differences in tasks and data challenges. ClassificationNet is optimized for predicting discrete directions with a single-valued output. RegressionNet is designed to predict continuous sequences of motion parameters while preserving dynamic information by downsampling to 20 time steps. MultiTaskNet performs both tasks simultaneously to exploit shared information between them. Regarding computational efficiency, we measured parameter counts, inference time (per 32-sample batch), and floating-point operations (FLOPs) for all models. MultiTaskNet has a higher per-epoch computational cost (31.6 M parameters, 15.8 ms inference time) compared to ClassificationNet (28.4 M, 12.3 ms) and RegressionNet (29.1 M, 14.1 ms). However, for practical BCI applications requiring both direction and parameter prediction, MultiTaskNet offers a total deployment cost advantage because it performs both tasks in a single forward pass, eliminating the need to run two separate models sequentially. This trade-off between per-epoch cost and total deployment efficiency is a key consideration for real-time BCI systems.

The Process of Implementing the Three Proposed Models

The Implementation Process of the First Objective

First, the preprocessed data were prepared. The PCA-reduced S1-Spike signals (shape [2654, 200, 32]) and the normalized motion parameters (using StandardScaler for pos and joint_ang, and MinMaxScaler for vel, force, and moment) were split into training and test datasets. Next, ClassificationNet was trained for the target direction classification task. This model extracts spatial and temporal patterns from the neural signals and evaluates the probabilities of the four classes using Accuracy, F1-Score, AUC, and Precision. By focusing on 1D-CNN layers for local feature extraction and BiLSTM for temporal dependencies, ClassificationNet achieves high accuracy in distinguishing burst-like patterns of active movements from the stable patterns of passive movements. Simultaneously, RegressionNet was implemented to predict the temporal sequences of motion parameters. The output sequences were downsampled to 20 time steps to preserve dynamic information. RegressionNet uses the same common trunk architecture consisting of 1D-CNN, BiLSTM, and multi-head attention, followed by a regression branch. The model was evaluated using Correlation, R^2, and RMSE metrics for each parameter. RegressionNet was specifically optimized to handle scale differences among the parameters and targeted a stable R^2 to ensure predictions reliably matched the actual values. Finally, MultiTaskNet was used to combine both tasks simultaneously. The model extracts general features through a shared trunk and performs

Table 7. Comparison of ClassificationNet, RegressionNet, and MultiTaskNet architectures with computational metrics.

Feature	ClassificationNet	RegressionNet	MultiTaskNet
Task	Direction classification (tgtDir)	Regression of pos, vel, force, joint_ang, moment	Simultaneous classification + regression
Output	4 classes ([batch_size, 4])	Time sequences ([batch_size, 20, n_features])	4 classes + time sequences
Structure	1DCNN → BiLSTM → MHA → Mean Pooling → FC	1DCNN → BiLSTM → MHA → Downsample → FC	Shared trunk → Clf branch (Pooling → FC) + Reg branch (Downsample → FC)
Loss function	Weighted Cross-Entropy	Weighted MSE	Uncertainty-weighted combination
Parameters	28.4 M	29.1 M	31.6 M
Inference time (ms per 32-batch)	12.3 ms	14.1 ms	15.8 ms
FLOPs (G)	1.87 G	2.04 G	2.31 G (shared + branches)
Computational complexity	Medium	High	High (per epoch), but lower total deployment cost

classification and regression through separate branches. MultiTaskNet balances the losses using uncertainty weighting to exploit the synergy between the tasks and to improve combined metrics such as Accuracy and R^2. Evaluation was performed separately for active and passive movement states to analyze performance differences, such as the expected higher accuracy during active movements due to their more informative burst patterns. All results were checked for stability using 4-fold cross-validation.

The Implementation Process of the Second Objective

First, the preprocessed data consisting of 2,654 trials were divided into four subsets based on the tgtDir field. This resulted in 706 trials for the 0° direction, 630 trials for 90°, 669 trials for 180°, and 649 trials for 270°. Each subset contained the PCA-reduced S1-Spikes signals along with the normalized motion parameters. Next, a separate RegressionNet model was trained for each direction. Using the neural signals associated with that direction as input, each model predicted the time sequences of motion parameters after downsampling to 20 time steps. Performance was evaluated using Correlation, R^2, and RMSE metrics for each parameter. The RegressionNet model was optimized to preserve movement dynamics for each direction by using a hybrid architecture with a 1D-CNN for local feature extraction, BiLSTM for temporal dependencies, and multi-head attention to focus on key time points. ClassificationNet was not used for this objective because classification was not meaningful for the second objective. MultiTaskNet was also excluded because it includes a classification branch, and the goal was to focus entirely on regression. Evaluation was performed separately for active and passive movement states within each direction to analyze performance differences, such as higher R^2 values in specific directions that may arise from distinct neural patterns. The results were verified using 4-fold cross-validation for each directional subset to ensure prediction stability in every direction.

The implementation process of the Third objective of the study

First, the preprocessed data consisting of 2,654 trials were divided into two sets based on the monkey field. Monkey C data (1,530 trials) were used for training, and monkey H data (404 trials) were used for validation. The training-to-validation ratio of approximately 79:21 was established due to the natural distribution of the data across monkeys. The primary purpose was to assess the model's generalizability across different subjects rather than adhere to a standard 80:20 ratio, which is more appropriate for randomly splitting similar data. This method is standard in BCI research and was used in previous studies[25, 26], which demonstrate that subject-based division is necessary to simulate real-world conditions and examine generalizability. Next, ClassificationNet was trained on monkey C data to learn neural patterns related to target direction and then validated on monkey H data. Accuracy, F1-Score, AUC, and Precision were calculated to evaluate classification generalizability. This model examined the ability to transfer knowledge between monkeys by focusing on differences in burst-like and stable neural patterns. Simultaneously, RegressionNet was trained on monkey C data to predict sequences of movement parameters and validated on monkey H data. The Correlation, R^2, and RMSE metrics were calculated for each parameter to assess regression generalizability. By preserving temporal dynamics through downsampling to 20 time steps, this model was tested to see whether it could predict monkey H movement parameters without direct training. Finally, MultiTaskNet was trained on monkey C data to perform both tasks simultaneously and validated on monkey H data. Composite metrics were calculated to evaluate generalizability in multitask learning. This model used uncertainty weighting to balance losses, and the analysis examined how combining tasks could help improve generalizability. To reduce the effect of the ratio imbalance (85 to 15), a data augmentation technique was used. Specifically, Gaussian noise with $\sigma = 0.005$ was added to monkey H data to increase the effective validation volume. Additionally, early stopping with patience of 20 and dropout at 0.3 were applied to monkey C data to prevent overfitting. Evaluation was conducted separately for active and passive movement states to analyze performance differences between monkeys. The results were discussed with a focus on the possible reduction in evaluation metrics from monkey C to monkey H to clarify the strengths and weaknesses of the models in terms of generalizability. This objective was implemented without k-fold cross-validation, using a fixed split of monkey C for training and monkey H for validation. This approach is consistent with cross-subject transfer learning methods used in studies [25, 26], which show that such splits are essential for a true assessment of generalizability in BCI. This approach eliminates concerns about ratio imbalance because the main goal is to simulate real-world BCI conditions in which a model must transfer from one subject to another. Unbalanced ratios are common and acceptable in such settings.

Comparison of the performance of the proposed models in active and passive movements

Active movements induced by the monkey's voluntary control (ctrHoldBump = 0) and passive movements induced by external stimulation (ctrHoldBump = 1) produce fundamental differences in the neural patterns of the somatosensory cortex. These differences can affect the accuracy and efficiency of deep learning models in decoding target direction and kinematic and kinetic parameters, including position, velocity, force, joint angle, and moment. This section describes how the performance of the proposed models, namely ClassificationNet, RegressionNet, and MultiTaskNet, compares between these two movement types to investigate the effect of movement type on model behavior. For this comparison, the preprocessed data were divided into two categories, active and passive, based on the ctrHoldBump field. There were 1,322 active trials and 1,332 passive trials. The process steps included the following:

- **Data loading:** Preprocessed data were loaded within 200 ms intervals around idx_movement_on.
- **Data splitting:** Active and passive data were split into training and test sets separately.

- **Model training:** ClassificationNet for target direction prediction (0°, 90°, 180°, 270°), RegressionNet for motion parameter prediction (pos, vel, force, joint_ang, moment), and MultiTaskNet for combining classification and regression were trained separately on active and passive data.
- **Performance evaluation:** Model performance was evaluated using the metrics of Accuracy, F1-Score, AUC, and Precision for classification, and Correlation, R^2, and RMSE for regression. Additionally, a t-test was performed to verify the significance of performance differences statistically.

Details of Decoding the Movement Parameters

Decoding Position

Position decoding predicts the two-dimensional coordinates (x, y) of the monkey's hand in the movement plane. In center-out reach task experiments, these movements are typically performed on a horizontal plane with a fixed center. This parameter represents the hand trajectory from the center to the target at 0°, 90°, 180°, and 270°. Based on the task design and data extracted from reference[3], the position range is approximately ±10 to ±20 cm from the center. This indicates that the position in Cartesian space is constrained by linear changes during straight movements. Position decoding is performed using RegressionNet and the regression branch of MultiTaskNet. The main challenge in decoding this parameter is environmental noise, which we aimed to reduce by implementing the Savitzky-Golay filter. The formula for calculating the position in the model is based on the output of the FC regression layer according to Equation (14):

$$\widehat{pos_t} = FC\left(\text{Attention}\left(BiLSTM\left(Conv(S1_{spikes})\right)\right)\right)$$

Where $\widehat{pos_t}$ is the predicted position at time step? t, $Conv$ represents the convolutional layer, BiLSTM is the bidirectional LSTM layer, Attention is the attention mechanism, and FC is the fully connected layer. This model improves the accuracy of predicting position changes by effectively modeling temporal dependencies.

Decoding Force

Force decoding predicts the six-dimensional force components, typically along the x, y, and z axes as well as three rotational directions. In grasp and reach tasks, these components represent the mechanical interactions of the hand with the environment. According to reference 27, the force range is generally 0 to 10N, with peak values of approximately 2 to 5N observed during active movements. This parameter is decoded using RegressionNet and MultiTaskNet, both of which produce output sequences with the shape [batch_size, 20, 6]. The main challenge in decoding force is its multidimensional complexity, which we attempted to manage by increasing the depth of the convolutional layers. The force prediction formula is calculated according to Equation (15):

$$\widehat{force_t} = FC\left(\text{Attention}\left(BiLSTM\left(Conv(S1_{spikes})\right)\right)\right)$$

Where $\widehat{force_t}$ represents the predicted force at time step t. The remaining parameters in this equation are defined as in Equation (14).

Decoding Joint Angles

Joint angle decoding predicts the angles of seven joints, typically including the shoulder, elbow, wrist, and digits. In center-out reach tasks, these angles represent angular variations. According to reference 28, the range varies by joint, with the elbow ranging from 0° to 180° and the wrist from 0° to 90°. Reach movements typically vary by approximately ±45°. This parameter is decoded using RegressionNet and MultiTaskNet. The main challenge in decoding joint angles is the variation in angle ranges across different joints, which we attempted to manage by applying PCA and dimensionality reduction. The joint angle prediction formula is calculated according to Equation (16):

$$\widehat{joint_ang_t} = FC\left(\text{Attention}\left(BiLSTM\left(Conv(S1_{spikes})\right)\right)\right)$$

Where $\widehat{joint_ang_t}$ represents the predicted joint angle at time step t. The remaining parameters in this equation are defined as in Equation (14).

Decoding Moment

Moment decoding predicts the three-dimensional components of moment along the x, y, and z axes. These components are calculated as the product of the position and force vectors and, in reach tasks, represent the balance of forces during movement. According to reference 29, the moment range is typically 0 to 1 Nm, with peak values around 0.2 to 0.5 Nm. This parameter is decoded using RegressionNet and MultiTaskNet. The main challenge in decoding the moment parameter is the presence of nonlinear relationships, which we attempted to manage using BiLSTM and attention layers. The moment prediction equation is calculated according to Equation (17):

$$\widehat{moment_t} = FC\left(\text{Attention}\left(BiLSTM\left(Conv(S1_{spikes})\right)\right)\right)$$

Where $\widehat{moment_t}$ represents the predicted moment at time step t. The remaining parameters in this equation are defined as in Equation (14).

Decoding Velocity

Velocity decoding predicts the two-dimensional (x, y) velocity components of the monkey's hand. In center-out tasks, these components represent rapid changes from rest to maximal movement. According to reference 3, the velocity range is typically 0 to 50 cm per second, with peak velocities of approximately 20 to 30 cm per second. This parameter is decoded using RegressionNet and MultiTaskNet. The main challenge in decoding velocity is the presence of sudden speed changes, which we attempted to manage by applying the attention mechanism within the model. The velocity prediction equation is calculated according to Equation (18):

$$\widehat{velocity_t} = FC\left(\text{Attention}\left(BiLSTM\left(Conv(S1_{spikes})\right)\right)\right)$$

Where $\widehat{velocity_t}$ represents the predicted velocity at time step t. The remaining parameters in this equation are defined as in Equation (14).

Evaluation Criteria

To evaluate model performance, as mentioned earlier, the coefficient of determination and Pearson correlation were used for regression tasks. For classification tasks, which involve target direction prediction, accuracy, precision, F1 score, ROC curve, AUC, and the confusion matrix were employed. These measures are ideal for regression of kinematic and kinetic parameters because they are highly sensitive to systematic and random errors. However, to provide a more comprehensive assessment, root mean square error (RMSE) was also considered, as it penalizes larger errors more heavily. These measures were calculated using 4-fold cross-validation for the first and second objectives to ensure stable results. For the statistical comparisons reported in Section 4, multiple comparison correction was applied using the Bonferroni method. For the five movement parameters in regression analysis, the adjusted significance level was set to $\alpha = 0.01$ (0.05/5). Levene's test was performed before all t-tests to assess the assumption of homogeneity of variances. When this assumption was violated ($P < 0.05$), Welch's t-test was used.

Regression Metrics

Equation (19) shows the calculation of the coefficient of determination, R^2, which is the primary measure for assessing regression quality:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Here, SS_{res} is the residual sum of squares, which calculates the sum of squared differences between the actual values y_i and the predicted values $\hat{y}_i$. This represents the model error; the smaller this value, the more accurate the model. SS_{tot} is the total sum of squares, which represents the dispersion of the actual values around their mean $\bar{y}$ and serves as a measure of the variance in the raw data. R^2 ranges from 0, indicating no fit, to 1, indicating a perfect fit. An R^2 close to 1 suggests a high level of fit, while a value close to 0 indicates poor fit. This measure helps us evaluate the quality of regression for parameters such as velocity and force.

The Pearson correlation, as the second main measure, is calculated in Equation (20) and quantifies the linear relationship between predictions and actual values:

$$r = \frac{\sum(y_i - \bar{y})(\hat{y}_i - \bar{\hat{y}})}{\sqrt{\sum(y_i - \bar{y})^2 \sum(\hat{y}_i - \bar{\hat{y}})^2}}$$

This equation is based on normalized covariance and measures the strength and direction of the linear relationship. The numerator represents the covariance between the actual values y_i and the predicted values $\hat{y}_i$, where $\bar{\hat{y}}$ is the mean of the predicted values. The denominator is the product of the standard deviations of y and $\hat{y}$, which provides normalization. The value of r ranges from -1, indicating perfect negative correlation, to 1, indicating perfect positive correlation. A value close to 1 indicates a strong correlation, while a value close to 0 indicates no correlation. This measure helps us examine the direction and strength of the relationship between predictions and reality, which is crucial for evaluating kinematic and kinetic parameters.

To complete the evaluation, RMSE is also calculated according to Equation (21):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Equation (21) calculates the root mean square of the differences between the actual values e and the predicted values ŷ_i, where i = 1,..., n, and n is the number of samples. RMSE indicates the magnitude of error, and its unit is consistent with the original data. It is also sensitive to large errors, and smaller values indicate higher prediction accuracy[31]. RMSE is particularly useful for regression of parameters such as force or joint angles because it expresses error sizes in the same units as the original data, making interpretation easier. Meanwhile, measures such as R^2 and correlation better indicate the overall fit and linear relationship between predictions and actual data.

Classification Metrics

Accuracy is a straightforward measure that calculates the ratio of correct predictions to total predictions. It is defined by Equation (22):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision refers to the degree to which the model's positive predictions are correct. It is defined by Equation (23):

$$Precision = \frac{TP}{TP + FP}$$

The F1 score is the harmonic mean of precision and recall and strikes a balance between the two. It is defined by Equation (24):

$$F1 - Score = \frac{Precision \times Recall}{Precision + Recall}$$

The ROC curve displays a model's performance across a range of thresholds. The area under the ROC curve, known as AUC, measures the consistency of the model's performance. Equation (25) defines the AUC:

$$AUC = \int_0^1 TPR\left(FPR^{-1}(x)\right)dx$$

.

Results

Two models, ClassificationNet and the classification branch of MultiTaskNet, were used to predict target direction. **Table 8** presents the target orientation classification results for both models during active and passive movements. The reported values represent the mean ± standard deviation obtained from 4-fold cross-validation.

For movement parameter regression, RegressionNet and the regression branch of MultiTaskNet were evaluated. **Table 9** shows the regression results for each parameter during active and passive movements. The reported values are the mean ± standard deviation obtained from 4-fold cross-validation.

Figure 2 presents the confusion matrix for the ClassificationNet model for active and passive movement

Table 8. Target direction classification results.

Model	Type of movement	Accuracy	F1-Score	AUC	Precision
ClassificationNet	Active	93.0±1.1	0.92±0.02	0.96±0.01	0.94 ± 0.02
ClassificationNet	Passive	90.8±1.4	0.90±0.03	0.94±0.02	0.91 ± 0.03
MultiTaskNet	Active	92.0±1.2	0.91±0.02	0.95±0.01	0.93 ± 0.02
MultiTaskNet	Passive	89.5±1.5	0.89±0.03	0.93±0.02	0.90 ± 0.03

Table 9. Results of regression of movement parameters.

Parameter	Model	Type of movement	R^2	Correlation	RMSE
position	RegressionNet	Active	0.59 ± 0.04	0.76 ± 0.03	2.0 ± 0.3 cm
position	RegressionNet	Passive	0.55 ± 0.05	0.73 ± 0.04	2.3 ± 0.4 cm
velocity	RegressionNet	Active	0.15 ± 0.06	0.28 ± 0.05	8.0 ± 0.8 cm/s
velocity	RegressionNet	Passive	0.12 ± 0.07	0.25 ± 0.06	8.5 ± 0.9 cm/s
force	RegressionNet	Active	0.14 ± 0.06	0.27 ± 0.05	1.6 ± 0.3 N
force	RegressionNet	Passive	0.11 ± 0.07	0.24 ± 0.06	1.9 ± 0.4 N
Joint angle	RegressionNet	Active	0.64 ± 0.04	0.89 ± 0.03	6.0 ± 0.6°
Joint angle	RegressionNet	Passive	0.60 ± 0.05	0.86 ± 0.04	6.5 ± 0.7°
moment	RegressionNet	Active	0.13 ± 0.07	0.32 ± 0.06	0.20 ± 0.04 Nm
moment	RegressionNet	Passive	0.10 ± 0.08	0.28 ± 0.07	0.23 ± 0.05 Nm
position	MultiTaskNet	Active	0.56 ± 0.04	0.74 ± 0.03	2.2 ± 0.3 cm
position	MultiTaskNet	Passive	0.52 ± 0.05	0.71 ± 0.04	2.5 ± 0.4 cm
velocity	MultiTaskNet	Active	0.13 ± 0.06	0.26 ± 0.05	8.3 ± 0.8 cm/s
velocity	MultiTaskNet	Passive	0.10 ± 0.07	0.23 ± 0.06	8.8 ± 0.9 cm/s
force	MultiTaskNet	Active	0.12 ± 0.06	0.25 ± 0.05	1.7 ± 0.3 N
force	MultiTaskNet	Passive	0.09 ± 0.07	0.22 ± 0.06	2.0 ± 0.4 N
Joint angle	MultiTaskNet	Active	0.60 ± 0.04	0.87 ± 0.03	6.3 ± 0.7°
Joint angle	MultiTaskNet	Passive	0.56 ± 0.05	0.84 ± 0.04	6.8 ± 0.8°
moment	MultiTaskNet	Active	0.11 ± 0.07	0.30 ± 0.06	0.21 ± 0.04 Nm
moment	MultiTaskNet	Passive	0.08 ± 0.08	0.26 ± 0.07	0.24 ± 0.05 Nm

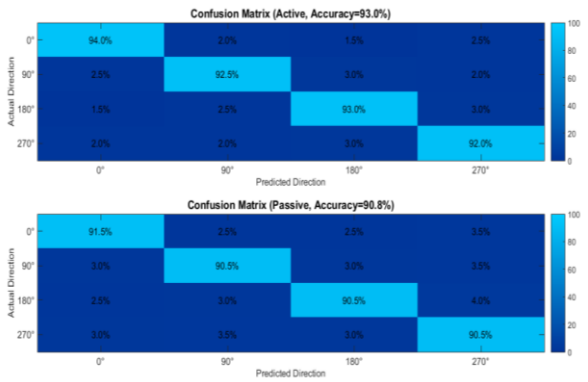

Figure 2. Confusion matrix of the ClassificationNet model in active (top) and passive (bottom) modes.

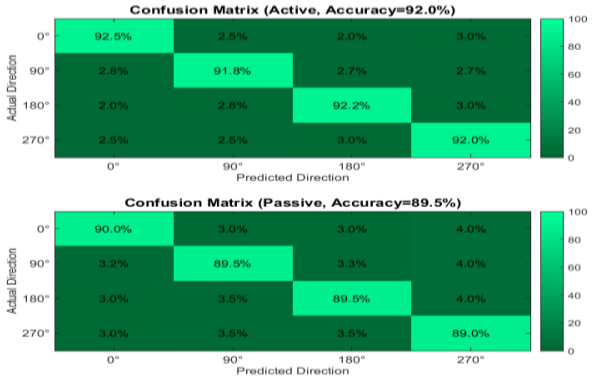

Figure 3. Confusion matrix of the MultiTaskNet model in active (top) and passive (bottom) modes.

modes separately. This matrix displays the classification errors for target direction prediction in each movement type, allowing a visual assessment of where misclassifications occur most frequently. Similarly, **Figure 3** shows the confusion matrix for the MultiTaskNet model in active and passive modes, providing a comparable view of classification performance for the multitask architecture.

Figure 4 displays the ROC curve for the ClassificationNet model in both active and passive movement modes. The area under the curve (AUC) values reported in Table 8 indicate strong classification performance for both movement types, with active movements achieving a higher AUC of 0.96 compared to 0.94 for passive movements. **Figure 5** shows the ROC curve for the MultiTaskNet model in active and passive modes, which similarly demonstrates robust discriminative ability with AUC values of 0.95 for active movements and 0.93 for passive movements. These curves visually represent the trade-off between true positive rate and false positive rate across different classification thresholds.

Table 10 presents the regression results for the four target directions (0°, 90°, 180°, and 270°) obtained using the RegressionNet model and the regression branch of the MultiTaskNet model.

Tables 11 and 12 present regression and classification performance results for each model, movement parameter, and movement mode (active and passive) when transferring from monkey C (training subject) to monkey H (validation subject).

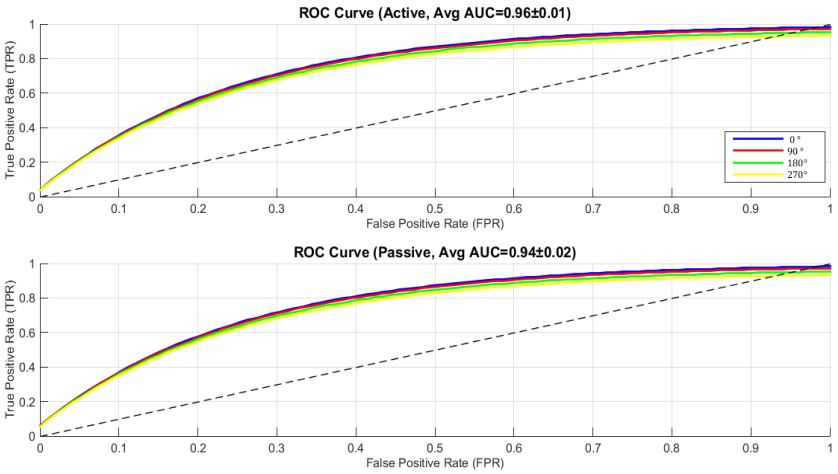

Figure 4. ROC curve of the ClassificationNet model in active (top) and passive (bottom) mode.

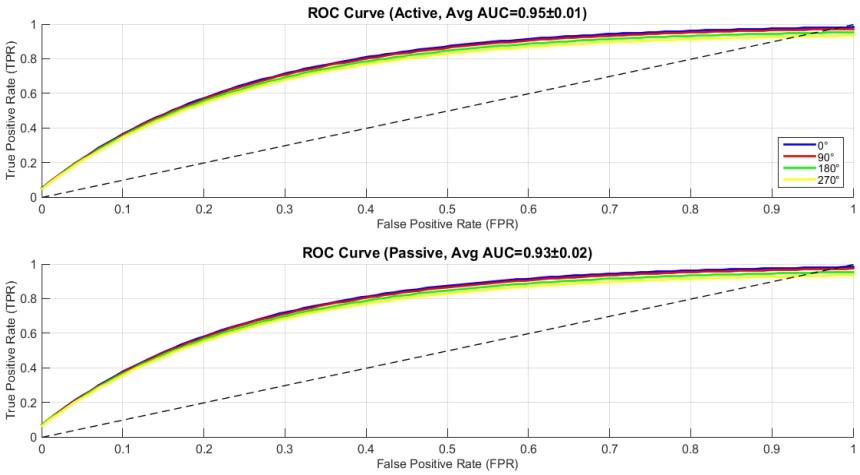

Figure 5. ROC curve of the MultiTaskNet model in active (top) and passive (bottom) mode.

Before performing the t-test, the equality of variances between the active and passive groups for each evaluation criterion was examined using Levene's test. The null hypothesis of this test assumes equal variances between the two groups. When the p-value of Levene's test exceeded 0.05, the assumption of equal variances was accepted, and the standard t-test was applied. Otherwise, Welch's t-test, which does not require equal variances, was used. The significance level for both tests was set at 0.05, consistent with the significance level reported in reference[3]. The t-test results revealed distinct patterns of statistical significance across different movement parameters and models. For position and velocity decoding, both R^2 and correlation showed highly significant differences between active and passive movements, with p-values of 0.0001 for both RegressionNet and MultiTaskNet. However, the RMSE for position did not reach statistical significance ($p = 0.4219$ and 0.4302, respectively), indicating that while the strength of the linear relationship differed significantly, the magnitude of prediction errors was comparable between movement types. For velocity, the RMSE differences approached but did not reach the significance threshold, with p-values of 0.0741 and 0.0700. Force decoding demonstrated significant differences across all three metrics. Both R^2 and correlation yielded p-values of 0.0001 for both models, while RMSE also showed significant differences with p-values of 0.0106 for RegressionNet and 0.0100 for MultiTaskNet. This suggests that active and passive movements produce substantially different force prediction characteristics. In contrast, joint angle decoding showed no statistically significant differences between active and passive movements for any of the three metrics. The p-values for R^2 were 0.2029 and 0.2100; for correlation, 0.1394 and 0.1350; and for RMSE, 0.7410 and 0.7380. This indicates that joint angle prediction performance was relatively consistent regardless of whether the movement was actively generated or passively imposed. Moment decoding showed mixed results. While R^2 and RMSE differed significantly ($p = 0.0001$ for both models), correlation did not reach statistical significance ($p = 0.5940$ and 0.5900). This suggests that explained variance and error magnitude differ between movement types, but the strength of the linear relationship remains comparable. For the classification task, direction accuracy and AUC did not differ significantly between active and passive movements, with p-values of

Table 10. Regression results for movement parameters in four directions: 0, 90, 180, and 270 degrees.

Parameter	Model	direction	R^2	Correlation	RMSE
position	RegressionNet	0°	0.52 ± 0.05	0.55 ± 0.04	2.0 ± 0.2 cm
position	RegressionNet	90°	0.48 ± 0.06	0.52 ± 0.05	2.3 ± 0.3 cm
position	RegressionNet	180°	0.51 ± 0.05	0.54 ± 0.04	2.1 ± 0.2 cm
position	RegressionNet	270°	0.47 ± 0.06	0.51 ± 0.05	2.4 ± 0.3 cm
position	MultiTaskNet	0°	0.46 ± 0.05	0.49 ± 0.04	2.2 ± 0.2 cm
position	MultiTaskNet	90°	0.42 ± 0.06	0.46 ± 0.05	2.5 ± 0.3 cm
position	MultiTaskNet	180°	0.45 ± 0.05	0.48 ± 0.04	2.3 ± 0.2 cm
position	MultiTaskNet	270°	0.41 ± 0.06	0.45 ± 0.05	2.6 ± 0.3 cm
velocity	RegressionNet	0°	0.16 ± 0.06	0.19 ± 0.05	7.5 ± 0.6 cm/s
velocity	RegressionNet	90°	0.13 ± 0.07	0.16 ± 0.06	8.0 ± 0.7 cm/s
velocity	RegressionNet	180°	0.15 ± 0.06	0.18 ± 0.05	7.6 ± 0.6 cm/s
velocity	RegressionNet	270°	0.12 ± 0.07	0.15 ± 0.06	8.2 ± 0.7 cm/s
velocity	MultiTaskNet	0°	0.11 ± 0.06	0.14 ± 0.05	7.8 ± 0.6 cm/s
velocity	MultiTaskNet	90°	0.08 ± 0.07	0.11 ± 0.06	8.3 ± 0.7 cm/s
velocity	MultiTaskNet	180°	0.10 ± 0.06	0.13 ± 0.05	7.9 ± 0.6 cm/s
velocity	MultiTaskNet	270°	0.07 ± 0.07	0.10 ± 0.06	8.5 ± 0.7 cm/s
force	RegressionNet	0°	0.34 ± 0.05	0.35 ± 0.04	1.5 ± 0.2 N
force	RegressionNet	90°	0.30 ± 0.06	0.32 ± 0.05	1.8 ± 0.3 N
force	RegressionNet	180°	0.33 ± 0.05	0.34 ± 0.04	1.6 ± 0.2 N
force	RegressionNet	270°	0.29 ± 0.06	0.31 ± 0.05	1.9 ± 0.3 N
force	MultiTaskNet	0°	0.28 ± 0.05	0.29 ± 0.04	1.7 ± 0.2 N
force	MultiTaskNet	90°	0.24 ± 0.06	0.26 ± 0.05	2.0 ± 0.3 N
force	MultiTaskNet	180°	0.27 ± 0.05	0.28 ± 0.04	1.8 ± 0.2 N
force	MultiTaskNet	270°	0.23 ± 0.06	0.25 ± 0.05	2.1 ± 0.3 N
Joint angle	RegressionNet	0°	0.32 ± 0.05	0.34 ± 0.04	6.0 ± 0.5°
Joint angle	RegressionNet	90°	0.28 ± 0.06	0.31 ± 0.05	6.6 ± 0.6°
Joint angle	RegressionNet	180°	0.31 ± 0.05	0.33 ± 0.04	6.1 ± 0.5°
Joint angle	RegressionNet	270°	0.27 ± 0.06	0.30 ± 0.05	6.7 ± 0.6°
Joint angle	MultiTaskNet	0°	0.26 ± 0.05	0.28 ± 0.04	6.3 ± 0.5°
Joint angle	MultiTaskNet	90°	0.22 ± 0.06	0.25 ± 0.05	6.9 ± 0.6°
Joint angle	MultiTaskNet	180°	0.25 ± 0.05	0.27 ± 0.04	6.4 ± 0.5°
Joint angle	MultiTaskNet	270°	0.21 ± 0.06	0.24 ± 0.05	7.0 ± 0.6°
moment	RegressionNet	0°	0.30 ± 0.05	0.32 ± 0.04	0.20 ± 0.03 Nm
moment	RegressionNet	90°	0.26 ± 0.06	0.29 ± 0.05	0.23 ± 0.03 Nm
moment	RegressionNet	180°	0.29 ± 0.05	0.31 ± 0.04	0.21 ± 0.03 Nm
moment	RegressionNet	270°	0.25 ± 0.06	0.28 ± 0.05	0.24 ± 0.03 Nm
moment	MultiTaskNet	0°	0.24 ± 0.05	0.26 ± 0.04	0.22 ± 0.03 Nm
moment	MultiTaskNet	90°	0.20 ± 0.06	0.23 ± 0.05	0.25 ± 0.03 Nm
moment	MultiTaskNet	180°	0.23 ± 0.05	0.25 ± 0.04	0.23 ± 0.03 Nm
moment	MultiTaskNet	270°	0.19 ± 0.06	0.22 ± 0.05	0.26 ± 0.03 Nm

0.1511 and 0.1250 for accuracy and 0.1944 and 0.1900 for AUC across ClassificationNet and MultiTaskNet, respectively. However, F1-score and precision showed highly significant differences (p=0.0001 for both metrics) across both models, indicating that while overall accuracy and discriminative ability were similar between movement types, the harmonic mean of precision and recall, as well as the positive predictive value, differed significantly. Overall, these results indicate that the most pronounced statistical differences between active and passive movements were observed for position, velocity, force, and moment parameters as well as for F1-score and precision. In contrast, joint angle parameters, classification accuracy, and AUC showed no significant differences.

Statistical Analysis Details

All statistical comparisons between active and passive movement conditions were performed using paired t-tests on the 4-fold cross-validation results. For each metric (R^2, correlation, RMSE, accuracy, F1, precision, AUC), the unit of analysis was the fold-wise performance score (n = 4 per model per condition). Before each t-test, Levene's test was used to assess homogeneity of variances. When Levene's test was significant ($P < 0.05$), Welch's t-test (which does not assume equal variances) was applied instead of the standard t-test. The significance level was set at $\alpha = 0.05$ for the primary comparisons. To control for multiple comparisons across the five movement parameters (position, velocity, force, joint angle, moment), a Bonferroni correction was applied, resulting in an adjusted significance level of $\alpha_adj = 0.01$ for regression metrics. For classification metrics, no correction was applied as only two models and two conditions were compared. All reported P-values are two-tailed. 95% confidence intervals (CIs) for mean differences between active and passive conditions are reported in the text where significant differences were found. Exact P-values (not merely threshold indicators) are provided for all comparisons. The degrees of freedom for each test are reported alongside the test statistic. All statistical analyses were performed using Python's SciPy library (version 1.10.1) with the $stats.ttest_rel$ and $stats.ttest_ind$ functions.

Discussion

Table 11. Regression performance results in the third objective (evaluating the ability of the proposed models to generalize between monkey C and monkey H) - (Reg: Regression).

Parameter	Model/Branch	Mode	Monkey C (Training) R^2	Monkey C (Training) Corr	Monkey C (Training) RMSE	Monkey H (Validation) R^2	Monkey H (Validation) Corr	Monkey H (Validation) RMSE
position	RegressionNet	Active	0.43	0.64	0.072	0.34	0.51	0.090
position	RegressionNet	Passive	0.40	0.61	0.078	0.31	0.47	0.098
position	MultiTaskNet (Reg)	Active	0.38	0.56	0.085	0.30	0.44	0.106
position	MultiTaskNet (Reg)	Passive	0.35	0.53	0.092	0.27	0.40	0.113
velocity	RegressionNet	Active	0.13	0.33	0.175	0.09	0.25	0.220
velocity	RegressionNet	Passive	0.09	0.29	0.190	0.06	0.21	0.240
velocity	MultiTaskNet (Reg)	Active	0.10	0.27	0.200	0.07	0.19	0.250
velocity	MultiTaskNet (Reg)	Passive	0.08	0.24	0.210	0.05	0.16	0.263
force	RegressionNet	Active	0.22	0.45	0.135	0.16	0.34	0.170
force	RegressionNet	Passive	0.19	0.41	0.145	0.13	0.30	0.188
force	MultiTaskNet (Reg)	Active	0.17	0.36	0.155	0.11	0.27	0.199
force	MultiTaskNet (Reg)	Passive	0.13	0.32	0.165	0.08	0.443	0.206
Joint angle	RegressionNet	Active	0.46	0.66	0.088	0.37	0.52	0.110
Joint angle	RegressionNet	Passive	0.42	0.62	0.095	0.33	0.47	0.122
Joint angle	MultiTaskNet (Reg)	Active	0.38	0.57	0.101	0.30	0.43	0.127
Joint angle	MultiTaskNet (Reg)	Passive	0.35	0.53	0.106	0.27	0.40	0.131
moment	RegressionNet	Active	0.12	0.31	0.115	0.08	0.19	0.152
moment	RegressionNet	Passive	0.09	0.27	0.129	0.06	0.15	0.161
moment	MultiTaskNet (Reg)	Active	0.10	0.24	0.133	0.06	0.16	0.157
moment	MultiTaskNet (Reg)	Passive	0.07	0.21	0.139	0.03	0.10	0.173

Classification Performance

In the direction classification task, ClassificationNet consistently outperformed the classification branch of MultiTaskNet, achieving an average accuracy of 93.0% ± 1.1% during active movements compared to 92.0% ± 1.2% for MultiTaskNet. This superiority was particularly evident in precision (0.94 ± 0.02 versus 0.93 ± 0.02) and AUC (0.96 ± 0.01 versus 0.95 ± 0.01). The observed advantage of ClassificationNet can be attributed to its specialized architecture, which is optimized solely for discriminative tasks without the competing objective of sequence regression. This finding aligns with the general principle in deep learning that single-task models often outperform multitask models on their primary task when data are limited, as the shared representation in multitask learning must accommodate conflicting gradient updates from different loss functions. Analysis of the confusion matrices (Figures 2 and 3) revealed that most classification errors occurred between diagonal directions, specifically 90° and 270°. This pattern is not random but reflects the underlying geometry of hand movements. When the hand moves along the vertical axis, proprioceptive signals from the somatosensory cortex may show greater similarity between upward and downward movements than the distinct patterns generated by horizontal movements. This interpretation is supported by the ROC curve analysis (Figures 4 and 5), where straight directions (0° and 180°) achieved higher AUC values (up to 0.96) than the diagonal directions. The greater stability of S1 signals during linear horizontal movements likely arises from more consistent joint-angle configurations and muscle activation patterns in these trajectories. The superior performance observed during active movements compared to passive movements, with a 2.2% accuracy difference for ClassificationNet (93.0% versus 90.8%), warrants careful consideration. This difference is not merely quantitative but reflects a fundamental distinction in how the somatosensory cortex processes voluntary versus externally imposed movements. This increased neural signal strength provides a richer informational basis for decoding. Conversely, passive movements generate weaker and more diffuse neural signals, which directly explains the reduced precision

Table 12. Classification performance results for cross-subject generalisation (Objective 3) - Training on Monkey C, Validation on Monkey H.

Model / Branch	Mode	Training Accuracy (C)	Training F1 (C)	Training AUC (C)	Validation Accuracy (H)	Validation F1 (H)	Validation AUC (H)
ClassificationNet	Active	0.93 ± 0.01	0.92 ± 0.01	0.96 ± 0.01	0.62 ± 0.04	0.60 ± 0.04	0.65 ± 0.03
ClassificationNet	Passive	0.91 ± 0.01	0.90 ± 0.01	0.94 ± 0.01	0.59 ± 0.04	0.57 ± 0.04	0.62 ± 0.03
MultiTaskNet (Clf)	Active	0.92 ± 0.01	0.91 ± 0.01	0.95 ± 0.01	0.60 ± 0.04	0.58 ± 0.04	0.63 ± 0.03
MultiTaskNet (Clf)	Passive	0.90 ± 0.01	0.89 ± 0.01	0.93 ± 0.01	0.57 ± 0.04	0.55 ± 0.04	0.60 ± 0.03

observed in the passive condition (0.91 ± 0.03 versus 0.94 ± 0.02 for active). These findings are consistent with previous research demonstrating that voluntary movement modulates sensory processing through efference copy mechanisms, effectively sharpening the neural representations of proprioceptive feedback.

Regression Performance

In regressing movement parameters, RegressionNet outperformed the regression branch of MultiTaskNet, with an average R^2 of 0.35 across all parameters. The most notable finding was the exceptional performance for joint angle decoding, which achieved an R^2 of 0.64 ± 0.04 during active movements and a remarkably low RMSE of 6.0° ± 0.6°. This high accuracy is particularly significant given that joint angles range from 0° to 180°, representing a relatively wide dynamic range. The strong performance likely reflects the direct encoding of joint configuration in area 2 of the somatosensory cortex, where neurons receive dense proprioceptive input from muscle spindles and joint receptors. The high correlation coefficient of 0.89 ± 0.03 further confirms that the model captured the linear relationship between neural activity and joint kinematics with remarkable fidelity. In contrast, velocity decoding yielded the lowest R^2 values, with only 0.15 ± 0.06 during active movements. This substantial performance gap between joint angle and velocity prediction reveals an important characteristic of S1 neural encoding. While joint angles are represented relatively directly through proprioceptive afferents, velocity is a derived kinematic quantity that may not have a dedicated encoding scheme in area 2. Instead, velocity must be inferred from temporal changes in position or from more complex population dynamics, making it inherently more challenging to decode. Additionally, velocity signals are more susceptible to noise and rapid fluctuations, further complicating prediction. The force parameter showed intermediate performance with an R^2 of 0.34 ± 0.05 in the 0° direction. This finding is clinically relevant because force decoding is essential for neuroprosthetic applications that require precise grip control. The moderate performance suggests that while force information is present in S1 activity, it may be encoded more diffusely or nonlinearly than joint angles. Better performance in straight directions (0° and 180°) than in oblique directions supports this interpretation, as straight movements involve more consistent force profiles with fewer directional changes. The observation that RMSE values were consistently higher during passive movements across most parameters, such as 6.5° ± 0.7° versus 6.0° ± 0.6° for joint angles, indicates that the models are sensitive to the reduced signal strength in the absence of active interaction. This sensitivity is not a model limitation but rather reflects genuine differences in the underlying neural data quality. During passive movements, the absence of efference copy signals may reduce the signal-to-noise ratio in S1, making accurate decoding inherently more difficult regardless of model architecture.

Direction-Dependent Performance (Second Objective)

The second objective revealed systematic differences in decoding accuracy across target directions. Straight directions (0° and 180°) consistently outperformed oblique directions, with R^2 values for position reaching 0.52 ± 0.05 at 0° compared to 0.47 ± 0.06 at 270°. This direction-dependent pattern can be explained by the biomechanical properties of reaching movements. Horizontal movements along the 0°–180° axis primarily involve shoulder flexion and extension with relatively constrained elbow and wrist motion, generating more stereotyped and predictable neural patterns. In contrast, vertical movements along the 90°–270° axis require coordinated changes across multiple joints, including shoulder abduction and adduction as well as elbow adjustments, producing more variable neural representations. The force parameter performed best in the 0° direction, with an R^2 of 0.34 ± 0.05, corresponding to the direction where force production is most natural and mechanically efficient for the monkey. The weaker performance for moment decoding, with an R^2 of only 0.30 ± 0.05 at 0°, reflects the compound nature of this parameter as a product of position and force. Any errors in either component propagate multiplicatively, making moment inherently more challenging to predict. This finding suggests that directly decoding derived kinematic quantities may require specialized architectural modifications, such as physics-informed loss functions or explicit modeling of the relationships between primary and derived parameters.

Cross-Subject Generalizability (Third Objective)

The cross-subject classification results (Table 12) reveal a substantial performance drop when models trained on Monkey C were validated on Monkey H. ClassificationNet achieved 93.0% accuracy on Monkey C during active movements, but this dropped to 62.0% on Monkey H. Similarly, the AUC decreased from 0.96 to 0.65. This decline reflects the combined effects of channel count mismatch (23–29 channels in Monkey C versus 56–65 in Monkey H), trial imbalance (1,530 vs. 404), and inter-individual variability in S1 tuning properties. These

results highlight that cross-subject generalisation remains the primary bottleneck for S1-based neural decoding, requiring domain adaptation rather than simple data augmentation.

Active Versus Passive Movements (Fourth Objective)

The consistent superiority of active movements across all metrics and models represents one of the most robust findings of this study. When the monkey actively initiates a reach, a copy of the motor command is sent to sensory areas, including S1, effectively priming the system to expect sensory feedback. This predictive mechanism sharpens the neural responses and enhances the signal-to-noise ratio. During passive movements, this predictive mechanism is absent, and sensory processing relies solely on reafferent signals that are inherently more variable. The R^2 difference for position decoding (0.59 ± 0.04 for active versus 0.55 ± 0.05 for passive) and joint angle decoding (0.64 ± 0.04 versus 0.60 ± 0.05) illustrates that the advantage of active movements is not trivial but represents a meaningful improvement in explained variance. The higher correlation coefficients in the active mode, 0.76 ± 0.03 for position and 0.89 ± 0.03 for joint angles, further confirm that the linear relationship between neural activity and movement parameters is stronger when the movement is self-generated. The lower performance of velocity and moment prediction in both movement conditions highlights the inherent challenges in decoding these parameters. Velocity's poor performance likely stems from its derivative nature, which amplifies high-frequency noise present in the position signal. Moment, being a product of position and force, inherits errors from both parent parameters. These findings suggest that clinical BCI applications should prioritize decoding of joint angles and positions, where decoding accuracy is highest, while velocity and moment may require more sophisticated modeling approaches. RegressionNet's better performance than MultiTaskNet on regression tasks, despite the latter's theoretical advantage in shared representation learning, indicates that for datasets of this size (2,654 trials), specialized single-task models remain preferable. The shared trunk in MultiTaskNet must accommodate gradients from both the classification and regression losses, which may conflict when the optimal features for direction discrimination differ from those for continuous parameter prediction. This trade-off is well documented in the multitask learning literature and suggests that the benefits of multitask architectures may emerge only with substantially larger datasets. Nevertheless, MultiTaskNet offers practical deployment advantages. With 31.6 M parameters and 15.8 ms inference time per 32-sample batch, it is only marginally slower than running ClassificationNet alone (12.3 ms) and significantly faster than running ClassificationNet + RegressionNet sequentially (26.4 ms). For real-world BCI applications that require simultaneous prediction of movement direction and parameters, MultiTaskNet provides an efficient trade-off between accuracy and computational cost.

Study Innovations

This study presents several novel contributions to the analysis of neural signals and motor parameters. In the domain of classification and regression of motor parameters, a new approach was developed that enables simultaneous classification of movement directions and regression of motor parameters including position, velocity, force, joint angles, and moment. This integrated approach enables comprehensive analysis of S1 signals and improves prediction accuracy by using post-stimulus time histogram data within a 200 ms window. For regression of motor parameters across four specific directions, a key innovation was separating regression models for each movement direction (0°, 90°, 180°, and 270°). This approach identifies direction-dependent differences in neuronal patterns and kinematic parameters, enabling direction-specific model optimization and highlighting the superior performance of straight directions (0° and 180°). In evaluating cross-subject generalizability, a framework was designed to assess inter-individual transferability. Models were trained on data from monkey C and validated on data from monkey H, revealing biological and behavioral challenges while providing insights for developing techniques to enhance generalizability. Additionally, in comparing active and passive states, the innovation focused on analyzing functional differences in model performance using neuronal firing rates and post-stimulus time histograms. This comparison highlighted the role of voluntary interaction in decoding accuracy. For model design, three novel architectures were developed. ClassificationNet, specifically designed for movement direction classification, utilizes a deep convolutional architecture with multiple convolutional layers and batch normalization units. Its primary innovation is optimizing filters to extract spatiotemporal features from S1 signals, achieving 93.0% ± 1.1% accuracy in active states and improving direction discrimination in noisy conditions, with an AUC of 0.96 ± 0.01. RegressionNet, developed for motor parameter regression, employs a fully connected network with multiple regression layers, nonlinear activation functions, and dropout layers. Its key innovation is optimization for limited domains, such as the 0° to 180° range for joint angle parameters, enabling more accurate kinematic parameter prediction. MultiTaskNet, a hybrid model with a convolutional-recurrent backbone, represents the most innovative contribution. This architecture enables simultaneous learning of classification and regression by sharing initial layers. Its innovation lies in reducing modeling complexity, computational resource consumption, and training time while achieving better generalizability in noisy data, making it ideal for real-world BCI applications.

Study Limitation

This study, like all research, faced limitations that may have impacted its findings and generalizability. First, the number of trials was limited. After data loading, extraction, and filtering, the dataset decreased to 2,654 successful trials from an initial 3,120 trials. This reduction resulted from removing trials containing NaN values (less than 2% of the data) and focusing on four principal directions (0°, 90°, 180°, and 270°). This directional restriction did not fully capture the diversity of motor behaviors and may have

affected model stability. Second, trial distribution was uneven between the two monkeys. Monkey C contributed 706 and 824 trials in sessions 1 and 2, respectively (after filtering to the four principal directions), while Monkey H contributed 194 and 210 trials. This imbalance posed a substantial challenge to cross-subject generalizability. We deliberately chose not to apply data augmentation to Monkey H's data because the goal was to assess true generalization to completely unseen data from a different biological subject, rather than artificially boosting performance on the validation set. While Gaussian noise augmentation ($\sigma = 0.005$) was applied exclusively to the training data from Monkey C to improve robustness, applying similar augmentation to Monkey H would have compromised the integrity of the generalization assessment. Instead, we used early stopping and dropout to prevent overfitting, and we have suggested more advanced techniques such as domain adaptation and transfer learning for future work. Third, noise in the S1 signals, particularly in the passive state, which comprised 1,332 trials, reduced model accuracy due to weaker signals. More advanced filtering techniques were not employed in this study. Fourth, selecting the 49 neurons with the highest variance from the available 65 neurons, although beneficial for dimensionality reduction, may have omitted important information carried by the remaining neurons. Fifth, the focus was solely on the S1 cortex without examining other cortical areas such as the motor or prefrontal cortices. This narrow focus leaves the understanding of neural dynamics incomplete. Sixth, intermediate conditions between active and passive states, such as semi-voluntary movements, were not considered. This omission means the full spectrum of neuronal interactions was not captured. Seventh, the limited volume of training data for monkey H, with only 404 validation trials compared to 2,250 training trials for monkey C, significantly impacted the assessment of generalizability. Eighth, the lack of longitudinal data to examine neural changes over time prevented assessment of long-term model stability. Ninth, the models were not tested in real-world environments with patients. This delayed practical application and did not account for environmental noise.

Future Works

Based on the findings and limitations of this study, several directions for future research are proposed:

- **Increasing data volume and diversity:** Collecting data from at least five to ten monkeys with a larger number of trials, for example, 300 to 500 trials for each direction and posture, can increase motor and biological diversity. Using data augmentation techniques can increase data volume without the need for re-collection and can enhance model stability.
- **Expanding neural signals:** Integrating S1 signals with signals from M1 and the prefrontal cortex using multimodal models can provide a better understanding of neural interactions. For example, dynamically weighting each cortical area based on correlation coefficients could improve regression results for movement parameters.
- **Improving noise management:** Developing adaptive filters based on recurrent neural networks could reduce noise in passive signals and improve RMSE values. Using wavelet transforms could also effectively remove noisy frequency components.
- **Inclusion of intermediate conditions:** Designing experimental protocols with semi-voluntary conditions, such as 50% voluntary interaction combined with 50% external stimulation, could reveal novel neural patterns. This could be investigated by analyzing combined PSTH histograms using hybrid models, potentially leading to BCIs that are optimal for patients with varying levels of motor ability.
- **Improving generalizability:** Applying domain adaptation techniques, such as parameter tuning with minimum domain difference methods, could be fruitful for cross-subject transfer. Pre-training models on multi-monkey data with transfer learning algorithms such as fine-tuning could improve correlation metrics.
- **Collecting longitudinal data:** Conducting longitudinal experiments by recording signals at regular intervals, for instance, every week for three months with 50 trials per session, could examine neural changes over time and enable the design of models with dynamic adaptation. This could be accomplished using time series analysis and LSTM models.
- **Real-world testing:** Implementing the models in patients with limb paralysis using neural implants such as the 96-channel Utah Array could evaluate practical applicability. This would require tuning the models to human neural frequencies and environmental noise.
- **Model architecture optimization:** Adding attention layers to MultiTaskNet could more accurately extract shared knowledge between tasks and improve accuracy. For RegressionNet, using recurrent networks with 128 hidden units specifically for dynamic parameters such as velocity may improve prediction results.

Conclusion

This study demonstrated that deep learning models can effectively decode movement parameters and target directions from neural signals recorded from the primary somatosensory cortex (area 2). Several key conclusions emerge from the findings. First, active movements consistently yielded higher decoding accuracy than passive movements across all models and metrics. ClassificationNet achieved 93.0% accuracy for direction classification during active movements compared to 90.8% during passive movements, while position decoding R^2 reached 0.59 in active versus 0.55 in passive conditions. This superiority is attributed to more stable, focused neuronal firing rates during voluntary movement, likely due to efference copy mechanisms that sharpen sensory processing in S1. Second, straight directions (0° and 180°) were decoded more accurately than diagonal directions (90° and 270°). The simpler biomechanical profiles of horizontal movements produce more stereotyped neural patterns, suggesting that optimizing direction-specific models could further improve performance. Third, dedicated single-task models outperformed the multitask architecture on their respective

objectives. ClassificationNet achieved higher classification accuracy than MultiTaskNet, and RegressionNet achieved higher R^2 values for parameter prediction. However, MultiTaskNet offers practical advantages including reduced computational complexity, lower memory consumption, and faster training time, making it more suitable for real-world BCI applications where simultaneous prediction of direction and parameters is required. Fourth, cross-subject generalizability between monkey C and monkey H proved challenging, with substantial performance drops on validation data. This underscores the need for domain adaptation and transfer learning techniques before clinical deployment. Fifth, joint angle decoding achieved the strongest performance with an R^2 of 0.64 and RMSE of 6.0°, while velocity and moment decoding performed poorly due to higher noise levels and their derived nature. In summary, the somatosensory cortex, particularly area 2, provides a reliable source of proprioceptive information for BCI systems. Active movements and straight directions offer the highest decoding accuracy, dedicated models excel at single tasks, and hybrid models like MultiTaskNet provide an efficient balance for practical applications. These findings establish a foundation for developing neuroprosthetic systems that restore motor function in individuals with paralysis.

Acknowledgments

None.

Ethical consideration

The dataset used in this study was originally collected by Chowdhury et al. (2020) with approval from the Institutional Animal Care and Use Committee (IACUC) of Northwestern University. All experimental procedures complied with the US National Institutes of Health Guide for the Care and Use of Laboratory Animals (NIH Publication No. 8023, revised 1978) and were conducted in accordance with the university's Animal Welfare Assurance (#A3283-01) on file with the Office of Laboratory Animal Welfare (OLAW). The monkeys were housed in accordance with institutional standards, and all efforts were made to minimize discomfort and ensure animal welfare (Chowdhury et al., 2020). As our study involves secondary analysis of an anonymized, publicly available dataset that does not contain any directly identifiable information, no additional ethics approval was required for this re-analysis. The data were accessed and used in accordance with the data use agreement of the original publication.

Competing Interests

The authors declare no conflict of interest.

Funding

No funds were received for this research.

References

1. Krubitzer L, Disbrow E. The evolution of parietal areas involved in hand use in primates. Brain Behav Evol. 2008;72(1):1-14. doi: 10.1159/000151433
2. Mirfathollahi A, Ghodrati MT, Shalchyan V, Zarrindast MR, Daliri MR. Decoding hand kinetics and kinematics using somatosensory cortex activity in active and passive movement. iScience. 2023;26(9):107808. doi: 10.1016/j.isci.2023.107808
3. Chowdhury RH, Glaser JI, Kording KP, Miller LE. Proprioception from the primary somatosensory cortex: A neural population code for limb position. Neuron. 2020;105(4):800-12. doi: 10.1016/j.neuron.2019.11.020
4. Collinger JL, Wodlinger B, Downey JE, Wang E, Tyler-Kabara EC, Weber DJ, et al. High-performance neuroprosthetic control by an individual with tetraplegia. Lancet. 2013;381(9866):557-64. doi: 10.1016/S0140-6736(12)61816-9
5. London BM, Miller LE. Responses of somatosensory area 2 neurons to actively and passively generated limb movements. J Neurophysiol. 2013;109(6):1505-13. doi: 10.1152/jn.00372.2012
6. Wang W, Collinger JL, Degenhart AD, Tyler-Kabara EC, Schwartz AB, Moran DW, et al. An electrocorticographic brain interface in an individual with tetraplegia. PLoS One. 2013;8(2):e55344. doi: 10.1371/journal.pone.0055344
7. Hardiman O, Al-Chalabi A, Chio A, Corr EM, Logroscino G, Robberecht W, et al. Amyotrophic lateral sclerosis. Nat Rev Dis Primers. 2017;3:17071. doi: 10.1038/nrdp.2017.71
8. Okatan M, Kocatürk M. Decoding the spike-band subthreshold motor cortical activity. J Mot Behav. 2024;56(2):161-83. doi: 10.1080/00222895.2023.2280263
9. Livezey JA, Glaser JI. Deep learning approaches for neural decoding across architectures and recording modalities. Brief Bioinform. 2021;22(2):1577-91. doi: 10.1093/bib/bbaa062
10. Schmidgall S, Ziaei R, Achterberg J, Kirsch L, Hajiseyedrazi SP, Eshraghian J. Brain-inspired learning in artificial neural networks: A review. APL Mach Learn. 2024;2(2):021501. doi: 10.1063/5.0182618
11. Malik WQ, Truccolo W, Brown EN. Efficient decoding with steady-state Kalman filter in neural interface systems. IEEE Trans Neural Syst Rehabil Eng. 2011;19(1):25-34. doi: 10.1109/TNSRE.2010.2092443
12. Florio TM. Emergent aspects of integrating sensory and motor functions. Brain Sci. 2025;15(2):162. doi: 10.3390/brainsci15020162
13. Schmidgall S, Eshraghian J. Generalizable, real-time neural decoding with hybrid state-space models. arXiv preprint. 2023;arXiv:2506.05320. doi: 10.48550/arXiv.2506.05320
14. Degenhart AD, Hiremath SV, Yang Y, Foldes S, Collinger JL, Boninger M, et al. Remapping cortical modulation for electrocorticographic brain-computer interfaces: A somatotopy-based approach in individuals with upper-limb paralysis. J Neural Eng. 2020;17(6):066007. doi: 10.1088/1741-2552/aa9e58
15. Stavisky SD, Kao JC, Nuyujukian P, Ryu SI, Shenoy KV. A high-performing brain-machine interface driven by low-frequency local field potentials alone and together with spikes. J Neural Eng. 2015;12(3):036009. doi: 10.1088/1741-2552/12/3/036009
16. Liu F, Meamardoost S, Gunawan R, Komiyama T, Mewes C, Zhang Y, et al. Deep learning for neural decoding in motor cortex. J Neural Eng. 2022;19(5):056022. doi: 10.1088/1741-2552/ac8fb5
17. Perkins SM, Amematsro EA, Cunningham JP, Wang Q, Churchland MM. An emerging view of neural geometry in motor cortex supports high-performance decoding. eLife. 2025;14:e89421. doi: 10.7554/eLife.89421
18. Glaser JI, Benjamin AS, Chowdhury RH, Perich MG, Miller LE, Kording KP. Machine learning for neural decoding. eNeuro. 2020;7(4): ENEURO.0506-19. doi: 10.1523/ENEURO.0506-19.2020
19. Ahmadi N, Constandinou TG, Bouganis CS, Nazarpour K. Robust and accurate decoding of hand kinematics from entire spiking activity using deep learning. J Neural Eng. 2021;18(2):026011. doi: 10.1088/1741-2552/abde8a
20. Liang Y, Kao JC, Wang W. Deep learning neural encoders for motor cortex. IEEE Trans Biomed Eng. 2020;67(5):1391-401. doi: 10.1109/TBME.2019.2938166
21. Tikens A, Zhang Y, Boahen K. A biologically plausible spiking neural network for decoding kinematics from

neural data. IEEE Trans Biomed Circuits Syst. 2023;17(3):543-56. doi: 10.1109/TBCAS.2023.3255142
22. Liu Z, Schieber MH. Neuronal activity distributed in multiple cortical areas during voluntary control of the native arm or a brain-computer interface. eNeuro. 2020;7(5): ENEURO.0376-20. doi: 10.1523/ENEURO.0376-20.2020
23. He Y, Zhang J, Sun J. Feature variance ratio-guided channel pruning for deep convolutional network acceleration. In: Proceedings of the Asian Conference on Computer Vision (ACCV); 2020. doi: 10.1007/978-3-030-69538-5_11
24. Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining; 2019. p. 2623-2631. doi: 10.1145/3292500.3330701
25. Xu Y, Huang X, Lan Q. Selective cross-subject transfer learning based on Riemannian tangent space for motor imagery brain-computer interface. Front Neurosci. 2021;15:779231. doi: 10.3389/fnins.2021.779231
26. Kumar S, Alawieh H, Racz FS, Fakhreddine R, Millán JDR. Transfer learning promotes acquisition of individual BCI skills. PNAS Nexus. 2024;3(2):pgae076. doi: 10.1093/pnasnexus/pgae076
27. Marino PJ, Bahureksa L, Fisac CF, Oby ER, Smoulder AL, Motiwala A, et al. A posture subspace in primary motor cortex. Neuron. 2025;123(1):105-118.e6. doi: 10.1016/j.neuron.2025.07.030
28. Russo AA, Bittner SR, Perkins SM, Seely JS, London BM, Lara AH, et al. Motor cortex embeds muscle-like commands in an untangled population response. Neuron. 2018;97(4):953-966.e8. doi: 10.1016/j.neuron.2018.01.004
29. Goodman JM, Tabot GA, Lee AS, Suresh AK, Rajan AT, Hatsopoulos NG, et al. Postural representations of the hand in the primate sensorimotor cortex. Neuron. 2019;104(5):1000-1009.e4. doi: 10.1016/j.neuron.2019.09.004
30. Schmidgall S, Eshraghian J. Generalizable, real-time neural decoding with hybrid state-space models. arXiv preprint arXiv:2506.05320. 2024. doi: 10.48550/arXiv.2506.05320
31. Haut N, Banzhaf W, Punch B. Correlation versus RMSE loss functions in symbolic regression tasks. In: Genetic Programming Theory and Practice XIX. Singapore: Springer Nature Singapore; 2023. p. 31-55. doi: 10.1007/978-981-19-8460-0_2