

Neural Networks for Temporal Pattern Recognition and Dynamic Arm Gesture Speed Estimation for Robot Control

Milán Zsolt Bagladi
hhpw8b@inf.elte.hu

Department of Artificial Intelligence
ELTE Eötvös Loránd University
Budapest, Hungary

László Gulyás
lgulyas@inf.elte.hu

Department of Artificial Intelligence
ELTE Eötvös Loránd University
Budapest, Hungary

Abstract

Deploying intelligent robotic systems that interact with humans through gestures requires neural networks capable of recognizing diverse temporal patterns. We present a systematic benchmark of ten abstract sequential tasks—five permutation-invariant (set) and five order-dependent (sequence) problems—evaluated across eighteen neural network architectures spanning recurrent, convolutional, attention-based, and set-function families. Beyond the core architecture–task grid, we explore numerous preprocessing and target-variable transformations, yielding more than 250 distinct experimental configurations. All variants are trained and tested under strictly identical conditions (fixed random seeds, shared hyperparameters, shared data splits) to ensure fair and reproducible comparison. Ranking across all ten tasks reveals four consistently top-performing architectures—BiGRU, TCN, Conv1D, and GRUReLU—all compact enough for real-time deployment (under 2 000 parameters in the benchmark setting). Based on this ranking, we apply three architecturally diverse top models (BiGRU, TCN, and GRUReLU) to a practical robotics problem: estimating the execution speed of dynamic arm gestures from skeletal keypoint sequences. Three speed interpretations (peak count, period time, and mean spike spacing) are evaluated on a custom dataset of eight traffic-related gesture classes comprising 256 710 frames recorded via OpenPose. The best configuration achieves a mean absolute error of 0.198 on the peak-count interpretation, corresponding to roughly 5% relative error, while the period-time interpretation reaches approximately 4% relative error, and the mean spike spacing interpretation approximately 8% relative error. These results demonstrate that neural networks can reliably estimate gesture speed from skeletal data, opening a path toward speed-aware gesture-controlled robotic systems.

CCS Concepts

• **Computing methodologies** → **Machine learning approaches**; **Computer vision**; Control methods; Supervised learning by regression; Supervised learning by classification.

Keywords

Sequential Pattern Recognition, Deep Learning Benchmark, Gesture Speed Estimation, Neural Architecture Comparison, Time Series Regression, Human–Robot Interaction

ACM Reference Format:

Milán Zsolt Bagladi and László Gulyás. 2026. Neural Networks for Temporal Pattern Recognition and Dynamic Arm Gesture Speed Estimation for Robot Control. In *Intelligent Robotics FAIR 2026 (IntRob '26)*, June 18–19, 2026, Budapest, Hungary. ACM, New York, NY, USA, 10 pages. <https://doi.org/10.1145/3831600.3831618>

1 Introduction

Human arm gestures are used to convey precise, safety-critical commands in a wide range of real-world scenarios: traffic officers regulate vehicle flow at intersections, railway signalers authorize train movements, and airport ground crew guide aircraft during taxiing and parking. In all these cases, automating the recognition of such gestures can improve safety and operational efficiency. In our prior work [3], we developed a real-time pipeline for recognizing the *type* of dynamic arm gestures using OpenPose keypoint extraction, a 1×1 normalization scheme, and a GRU-based classifier. That system achieved high accuracy across varying camera angles and gesture speeds, and was successfully demonstrated controlling a TurtleBot 3 mobile robot.

However, recognizing *what* gesture is performed is only part of the problem. In many practical scenarios, the *speed* at which a gesture is executed carries additional meaning. For example, a rapid circular arm motion may signal urgency, while a slow one indicates a routine command. Our prior pipeline was shown to be largely insensitive to speed variations during *classification*—an advantage for type recognition, but a limitation when speed itself must be measured.

The speed measurement problem can be formulated as follows: given a sliding window of skeletal keypoint frames depicting a cyclic gesture, estimate a scalar value that characterizes its execution speed. As shown in [3], this can be approached algorithmically by selecting a characteristic *starting position* (null position) for each gesture, computing the per-frame distance from this position, and measuring the number of frames between consecutive local minima of the resulting distance curve. This yields a discrete speed value (in frames per cycle) that can be directly linked to wall-clock time given the camera’s frame rate.

A natural question arises: can neural networks learn to perform such speed estimation directly from the raw distance curves, without manually written rules to detect local minima? More broadly,



This work is licensed under a Creative Commons Attribution 4.0 International License. *IntRob '26, Budapest, Hungary*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2435-0/26/06

<https://doi.org/10.1145/3831600.3831618>

which neural network architectures are best suited for recognizing abstract temporal patterns in sequential data?

To answer this question rigorously, we designed a benchmark of ten abstract tasks that isolate specific pattern-recognition capabilities required for speed estimation and related problems. Five tasks are *set-type* (permutation-invariant), where the order of elements does not affect the target value, and five are *sequence-type*, where temporal order is essential. We evaluate eighteen diverse neural network architectures on all ten tasks under strictly controlled, identical experimental conditions, rank the architectures by aggregated per-task performance, and apply the three top-ranked models to a real-world gesture speed estimation problem using three distinct speed interpretations on a custom dataset of eight traffic-related arm gestures. Our experiments show that neural networks can estimate gesture speed from skeletal data with relative errors ranging from 4% to 8% across three speed interpretations, validating the approach for practical robotics deployment.

The main contributions of this paper are:

- A systematic benchmark of 18 neural network architectures across 10 abstract sequential tasks (5 permutation-invariant, 5 order-dependent), with over 250 experimental configurations evaluated under strictly controlled, identical conditions.
- A rank-based aggregation methodology that enables fair cross-task architecture comparison despite heterogeneous output scales, yielding a reproducible global architecture ranking.
- Validation of the benchmark findings on a practical robotics problem: neural speed estimation of dynamic arm gestures from skeletal keypoint data, achieving relative errors at or below 5% on two of three speed interpretations.

The paper begins with a review of related work on gesture recognition and neural sequence modeling (Section 2), then introduces the benchmark task suite that captures the key pattern-recognition capabilities needed for speed estimation (Section 3). Section 4 presents the eighteen evaluated architectures, spanning recurrent, convolutional, attention-based, and set-function families. The experimental protocol and ranking methodology are detailed in Section 5, followed by the benchmark results and global architecture ranking in Section 6. Building on these findings, Section 7 applies the top-ranked models to real gesture speed estimation, and Section 8 concludes with a discussion of future directions.

2 Related Work

2.1 Gesture Recognition in Robotics

Recent studies combine keypoint extraction and deep learning for gesture recognition in traffic and robotics. Bagladi [2, 3] uses OpenPose with neural models; He *et al.* [7] combine a modified CPM, hand-crafted spatial features, and LSTM; and Zanfir *et al.* [15] enrich pose descriptors with joint speed and acceleration for low-latency action recognition. Vieira *et al.* [12] use speed-aware action graphs, and Stadelmayer *et al.* [10] pair classification with regression of continuous gesture attributes. Yet none estimates arm-gesture execution *speed* from skeletal keypoint data as a continuous scalar.

2.2 Neural Architectures for Sequential Data

The architectures evaluated in this work span several families. Recurrent models such as GRU [6] and LSTM [8] process sequences element by element and maintain hidden states that capture temporal dependencies. Bidirectional variants [9] (BiGRU, BiLSTM) process the sequence in both directions. Temporal Convolutional Networks (TCN) [4] use dilated causal convolutions to capture long-range dependencies with efficient parallel computation. Transformer-based models [11, 13] use self-attention to capture global dependencies. Set-function architectures such as DeepSets [14] are specifically designed for permutation-invariant problems. Our benchmark systematically compares representatives of all these families on a controlled set of tasks.

3 Benchmark Tasks

We define ten tasks that each test a specific capability of neural networks when processing fixed-length sequences of numbers. Each task takes as input a sequence $\mathbf{x} = (x_1, x_2, \dots, x_N)$ of N real values and requires the model to produce a single scalar output $\hat{y}$. The tasks are divided into two groups based on whether the ordering of elements affects the target value.

3.1 Set Tasks (Permutation-Invariant)

For these five tasks, the target value y depends only on the *multiset* of input values, not their order. These tasks are where order-independent architectures such as DeepSets and AttentionPool have a theoretical advantage, though as we will see, how well a model trains in practice matters just as much as its theoretical design. In all five tasks, the target outputs are uniformly distributed over their respective ranges, ensuring balanced training conditions.

- (1) **Decision:** Determine whether the binary sequence $(x_i \in \{0, 1\})$ contains at least one element equal to 1.

$$y = 1[\exists i : x_i = 1], \quad y \in \{0, 1\}.$$

This binary classification task is the simplest in the benchmark and serves as a sanity check.

- (2) **Counting:** Count the number of elements equal to 1 in a binary sequence $(x_i \in \{0, 1\})$.

$$y = \sum_{i=1}^N 1[x_i = 1], \quad y \in \{0, 1, \dots, N\}.$$

All possible counts are represented uniformly in the training data.

- (3) **Variance:** Estimate the variance of the input sequence. Elements are drawn from $\text{Uniform}(\mu - w, \mu + w)$, giving $\text{Var}(\mathbf{x}) = w^2/3$.

$$y = \frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2, \quad y \in [0, 0.25].$$

- (4) **Mean:** Estimate the arithmetic mean of elements drawn from $[0, 1]$.

$$y = \frac{1}{N} \sum_{i=1}^N x_i, \quad y \in [0, 1].$$

- (5) **Maximum:** Estimate the maximum value of the sequence.

$$y = \max(x_1, \dots, x_N), \quad y \in [0, 1].$$

3.2 Sequence Tasks (Order-Dependent)

For these five tasks, the target value depends on the *ordering* of the input elements. The temporal structure of the input carries essential information, placing models with explicit sequence-processing mechanisms—such as recurrent networks and causal convolutions—at a potential advantage.

- (6) **Speed:** Estimate the mean absolute consecutive difference of a sequence with values in $[0, 1]$.

$$y = \frac{1}{N-1} \sum_{i=1}^{N-1} |x_{i+1} - x_i|, \quad y \in [0, 0.5].$$

- (7) **Peak Count:** Count the number of local maxima in the sequence. A point x_i is a peak if $x_{i-1} < x_i > x_{i+1}$. Sequences are generated as AR(1) processes with varying autocorrelation to control peak density.

$$y = |\{i : x_{i-1} < x_i > x_{i+1}\}|, \quad y \in [0, N/2].$$

- (8) **Period Time:** Estimate the number of complete cycles of a sinusoidal signal $x_i = \sin(2\pi f \cdot i/N)$, where $f \in [1, 11]$ is the cycle count within the N -sample window.

$$y = f, \quad y \in [1, 11].$$

- (9) **Spike Spacing:** Estimate the mean temporal distance between spike events in a sparse signal. The sequence consists of a baseline (≈ 0) with 2–6 spikes (≈ 1) at random positions.

$$y = \frac{1}{K-1} \sum_{k=1}^{K-1} (p_{k+1} - p_k), \quad y \in [3, 30] \text{ frames},$$

where $p_1, \dots, p_K$ are the spike positions.

- (10) **Threshold Crossings:** Count the number of upward crossings of the threshold 0.5.

$$y = \sum_{i=1}^{N-1} 1[x_i < 0.5 \wedge x_{i+1} \geq 0.5].$$

Tasks 7, 8, and 9 (Peak Count, Period Time, and Spike Spacing) were specifically chosen because they correspond to three natural interpretations of gesture execution speed (see Section 7): counting how many complete gesture cycles occur in a window, measuring the duration of a single cycle, and computing the average inter-cycle distance. The remaining tasks complement these by testing the underlying skills needed—detecting the presence of events (Task 1), counting elements (Task 2), estimating statistical summaries (Tasks 3–5), measuring rate of change (Task 6), and detecting transitions (Task 10).

4 Neural Network Architectures

We evaluate eighteen architectures spanning five broad families. All models map an input sequence $\mathbf{x} \in \mathbb{R}^N$ to a single scalar output $\hat{y} \in \mathbb{R}$ (binary classification for Task 1). Each architecture is described below; Tables 1 and 2 list the corresponding parameter counts.

4.1 Recurrent Models

Six architectures process the sequence element by element, maintaining a hidden state that captures temporal dependencies. **GRU** [6] is a Gated Recurrent Unit whose final hidden state is mapped to the output via a linear layer; we test basic, higher-hidden-dimension,

and linear-activation variants. **GRUReLU** applies LayerNorm [1] followed by ReLU to the GRU’s final hidden state before the linear output layer, replacing the default linear projection and avoiding output saturation. **BiGRU** [9] is a bidirectional GRU that concatenates forward and backward hidden states, followed by LayerNorm and a linear output. **LSTM** [8] is the classic Long Short-Term Memory network, tested in the same three variants as GRU. **BiLSTM** is the bidirectional analogue of BiGRU using LSTM cells. **SkipGRU** augments the GRU with skip connections that add the input directly to the hidden state.

4.2 Convolutional Models

Two architectures use 1D convolutions to capture local and long-range patterns. **Conv1D** applies parallel convolution branches with kernel sizes 1, 3, and 5, followed by global average and max pooling. **TCN** [4] (Temporal Convolutional Network) uses dilated causal convolutions (dilation factors 1, 2, 4) with ReLU, global pooling, and auxiliary features (input mean, $\log N$). Conv1D can be viewed as a simplified, non-causal variant of TCN.

4.3 Attention and Transformer Models

Three architectures use attention-based aggregation. **Transformer** is a permutation-invariant model built from an element-wise MLP (φ), multi-aggregation pooling (mean, max, min, std, sum/ N), and a readout MLP (ρ). Unlike standard Transformer architectures, this variant employs *no self-attention* and no positional encoding; its name reflects the aggregation-based readout style it shares with certain Transformer decoder modules. It is functionally a learnable extension of DeepSets with richer pooling statistics. **Transformer-PosCoding** is identical but prepends sinusoidal positional encoding to each input element, breaking permutation invariance and making it order-aware. **AttentionPool** learns attention weights via a scoring network (Linear $\rightarrow$ tanh $\rightarrow$ Linear $\rightarrow$ softmax) and produces attended sum, max, mean, and sum aggregations.

4.4 Set-Function Models

Six architectures are designed around permutation-invariant primitives. **DeepSets** [14] applies an element-wise MLP (φ), mean/max/sum pooling, and a readout MLP (ρ); it is theoretically universal for permutation-invariant functions. **Histogram** learns soft bin assignments via Gaussian kernels with trainable centers, producing a fixed-size histogram processed by an MLP. **Quantile** computes 21 quantile points (0%, 5%, ..., 100%) of the input, yielding a fixed-size summary processed by an MLP. **LogSumExp** approximates smooth maximum/minimum via the log-sum-exp operator with learnable temperature parameters. **PowerMean** computes generalized power means $M_p(\mathbf{x}) = (\frac{1}{N} \sum x_i^p)^{1/p}$ with learnable exponents p . **ElemAgg** applies element-wise learnable thresholds and aggregates six features (sum, mean, max of both transformed and raw values).

4.5 Baseline

NN is a simple feedforward network Linear($N \rightarrow 1$) that treats the sequence as a flat input vector. It is *not* length-invariant and serves as a naïve baseline.

5 Experimental Setup

5.1 Data Generation

For each task, training and test data are generated synthetically with full control over the target distribution. A key design principle is that target values are *uniformly distributed* across the output range, preventing bias toward any particular output region. Sequences of fixed length $N = 50$ are generated with controlled parameters specific to each task, matching the sliding window used in the gesture speed estimation application (Section 7).

Multiple dataset size configurations were tested: $N_{\text{train}}:N_{\text{test}} \in \{50:50, 250:250, 1000:1000, 50:1000, 50:10000\}$, where N_{train} and N_{test} denote the number of training and test samples, respectively. Symmetric configurations ($N_{\text{train}} = N_{\text{test}}$) evaluate standard performance under matched conditions, while asymmetric ones (50:1000, 50:10000) test how well models generalize to a more densely sampled output range than they were trained on. The primary evaluation uses the 1000:1000 configuration. The asymmetric configurations confirmed consistent relative architecture rankings and are therefore not reported separately.

5.2 Fair Comparison Protocol

To ensure that observed performance differences reflect genuine architectural strengths rather than other influences, every experiment follows a strict fair-comparison protocol. The core grid comprises 18 architectures $\times$ 10 tasks, but for several architectures—notably GRU, LSTM, and SkipGRU—we additionally explore multiple preprocessing and target-variable transformations such as linear scaling, density-based encoding, hidden-dimension variants, and their combinations. This yields more than 250 distinct experimental configurations, documented in over 500 individual training and evaluation notebooks.

Across all configurations, three invariants are maintained. First, fixed random seeds guarantee that every architecture receives exactly the same data splits for each task, so that data variability cannot affect the comparison. Second, all models share identical hyperparameters: Adam optimizer with a learning rate of 0.01, batch size of 32, a maximum of 200 training epochs, and early stopping with a patience of 5 epochs. Third, evaluation metrics are uniform: Task 1 is assessed by accuracy (equivalently, error rate = $1 - \text{acc}$), while Tasks 2–10 are assessed by both Mean Absolute Error (MAE) and Mean Squared Error (MSE).

5.3 Ranking Methodology

Because each task operates on a different output scale and unit, raw MAE and MSE values are not directly comparable across tasks. A low MAE on one task may represent excellent performance while the same numerical value on another task may be poor. To obtain a fair cross-task comparison, we therefore adopt a rank-based aggregation strategy. For each task, all 18 architectures are sorted by their primary metric (MAE for Tasks 2–10, accuracy for Task 1) and assigned *dense ranks*: metric values are rounded to three decimal places before comparison, rank 1 is best, ties (equal rounded values) receive the same rank, and the next distinct value receives rank +1 (no gaps). Architectures that fail to produce valid output on a task

(e.g., diverged training) receive one rank below the last valid result on that task.

The *global rank sum* for each architecture is the sum of its per-task ranks: $\Sigma_{\text{rank}} = \sum_{t=1}^{10} r_t$, where r_t is the architecture's dense rank on task t . Lower Σ_{rank} values indicate more consistently strong performance across the full task suite.

6 Benchmark Results

6.1 Set Tasks (1–5)

Table 1 summarizes the performance of all architectures on the five set tasks using the 1000:1000 configuration. The metric shown is accuracy for Task 1 and MAE for Tasks 2–5.

Key observations on set tasks:

- **Task 1 (Decision)** is solved perfectly by 10 of 18 architectures, confirming it as an easy baseline. GRUReLU and Histogram are tied at rank 2 (both 0.994).
- **Task 2 (Counting)** creates the sharpest spread: DeepSets and LogSumExp are tied at rank 1 with MAE = 0.000, followed by ElemAgg (0.001, rank 2) and Transformer (0.006, rank 3). Recurrent models (LSTM, BiLSTM, SkipGRU) struggle with MAE > 12.
- **Task 3 (Variance)** is tied at rank 1 among GRU, GRUReLU, and BiGRU (all 0.004), followed by TCN (0.005).
- **Task 4 (Mean)** has a two-way rank-1 tie: Quantile and PowerMean (both 0.000), and a four-way rank-2 tie (GRU, BiGRU, TCN, Histogram, all 0.001).
- **Task 5 (Max)** is best solved by Transformer (0.003), followed by PowerMean (0.004) and Quantile (0.005).
- Notably, the order-independent architectures (DeepSets, Histogram, Quantile, LogSumExp, AttentionPool) do not uniformly outperform sequence-oriented ones on these set tasks, confirming that training behavior matters as much as theoretical design.

6.2 Sequence Tasks (6–10)

Table 2 shows results for the five sequence tasks.

Key observations on sequence tasks:

- **Task 6 (Speed)** is best solved by GRU (0.004), followed by TCN (0.005), then GRUReLU and Conv1D tied at rank 3 (both 0.006). Many architectures converge to MAE $\approx$ 0.084, indicating they learn only a constant-output baseline.
- **Task 7 (Peak Count)** strongly differentiates architectures: DeepSets (5.203), Transformer (6.794), and BiGRU (8.023) succeed, while most recurrent models fail entirely (MAE > 100).
- **Task 8 (Period Time)** is tied at rank 1 between Conv1D and TCN (both 0.030), with BiGRU (0.031) close behind. This task rewards architectures capable of detecting periodic structure.
- **Task 9 (Spike Spacing)** is dominated by TCN (4.894, rank 1). The remaining 17 models converge to $\approx$ 6.752–6.803, yielding a large tied block at the baseline mean prediction.
- **Task 10 (Threshold)** is relatively difficult for all architectures, with BiGRU (6.243) and GRUReLU (6.294) achieving the best results.

Table 1: Performance on set tasks (1000:1000 configuration). Task 1: accuracy ($\uparrow$); Tasks 2–5: MAE ($\downarrow$). Values rounded to 3 decimal places. Bold = rank 1. Superscript = per-task dense rank (ties share the same rank). – = convergence failure (ranked last).

Architecture	Params	1-Decision	2-Counting	3-Variance	4-Mean	5-Max
GRU	14	1.000	3.428 ¹¹	0.004	0.001 ²	0.006 ⁴
GRUReLU	963	0.994 ²	1.226 ⁹	0.004	0.002 ³	0.011 ⁹
BiGRU	1,921	1.000	2.020 ¹⁰	0.004	0.001 ²	0.007 ⁵
LSTM	16	0.960 ⁵	12.726 ¹⁴	0.025 ⁴	0.012 ⁷	0.020 ¹¹
BiLSTM	2,529	0.988 ⁴	14.126 ¹⁵	0.012 ³	0.015 ⁹	0.033 ¹³
SkipGRU	14	1.000	19.444 ¹⁶	0.062 ⁹	0.014 ⁸	0.044 ¹⁴
Conv1D	292	1.000	0.108 ⁵	0.012 ³	0.004 ⁴	0.009 ⁷
TCN	1,667	1.000	0.502 ⁷	0.005 ²	0.001 ²	0.009 ⁷
Transformer	24,961	1.000	0.006 ³	0.063 ¹⁰	0.004 ⁴	0.003
TransfPosCod	29,665	0.500 ⁶	3.890 ¹²	0.063 ¹⁰	0.250 ¹³	0.250 ¹⁵
AttentionPool	30	1.000	0.670 ⁸	0.225 ¹³	0.218 ¹²	0.029 ¹²
DeepSets	16,705	0.500 ⁶	0.000	0.044 ⁷	0.005 ⁵	0.013 ¹⁰
Histogram	406	0.994 ²	0.022 ⁴	0.042 ⁶	0.001 ²	0.009 ⁷
Quantile	385	1.000	12.537 ¹³	0.012 ³	0.000	0.005 ³
LogSumExp	12	1.000	0.000	0.060 ⁸	0.010 ⁶	0.010 ⁸
PowerMean	11	0.500 ⁶	–	0.129 ¹²	0.000	0.004 ²
ElemAgg	9	1.000	0.001 ²	0.028 ⁵	0.026 ¹¹	0.008 ⁶
NN	1,001	0.992 ³	0.266 ⁶	0.069 ¹¹	0.024 ¹⁰	0.010 ⁸

Table 2: Performance on sequence tasks (1000:1000 configuration, MAE $\downarrow$). Values rounded to 3 decimal places. Bold = rank 1. Superscript = per-task dense rank (ties share the same rank). – = convergence failure (ranked last).

Architecture	Params	6-Speed	7-Peak	8-Period	9-SpikeSp	10-Threshold
GRU	14	0.004	225.496 ¹⁸	2.500 ¹³	6.752 ²	10.318 ¹³
GRUReLU	963	0.006 ³	123.024 ¹¹	0.074 ⁴	6.752 ²	6.294 ²
BiGRU	1,921	0.008 ⁴	8.023 ³	0.031 ²	6.752 ²	6.243
LSTM	16	0.083 ⁹	225.267 ¹⁷	2.500 ¹³	6.753 ³	133.807 ¹⁶
BiLSTM	2,529	0.069 ⁷	113.481 ¹⁰	0.092 ⁵	6.752 ²	6.302 ³
SkipGRU	14	0.063 ⁶	180.412 ¹⁵	2.501 ¹⁴	6.759 ⁶	241.764 ¹⁷
Conv1D	292	0.006 ³	18.842 ⁵	0.030	6.754 ⁴	8.516 ¹⁰
TCN	1,667	0.005 ²	8.361 ⁴	0.030	4.894	7.163 ⁴
Transformer	24,961	0.073 ⁸	6.794 ²	1.923 ⁸	6.753 ³	10.346 ¹⁴
TransfPosCod	29,665	0.084 ¹⁰	180.538 ¹⁶	2.871 ¹⁵	6.774 ⁷	–
AttentionPool	30	0.119 ¹⁴	153.432 ¹³	1.979 ¹¹	6.798 ⁹	8.314 ⁶
DeepSets	16,705	0.084 ¹⁰	5.203	2.028 ¹²	6.754 ⁴	8.496 ⁹
Histogram	406	0.084 ¹⁰	20.789 ⁶	1.801 ⁶	6.753 ³	10.424 ¹⁵
Quantile	385	0.061 ⁵	123.444 ¹²	1.823 ⁷	6.752 ²	8.979 ¹²
LogSumExp	12	0.089 ¹²	107.060 ⁹	1.975 ¹⁰	6.756 ⁵	8.333 ⁸
PowerMean	11	0.087 ¹¹	153.812 ¹⁴	1.973 ⁹	6.803 ¹⁰	8.329 ⁷
ElemAgg	9	0.089 ¹²	72.846 ⁷	1.975 ¹⁰	6.775 ⁸	8.657 ¹¹
NN	1,001	0.094 ¹³	87.444 ⁸	0.034 ³	7.615 ¹¹	7.268 ⁵

6.3 Global Ranking

Table 3 summarizes the global rank sum Σ_{rank} for all architectures. **BiGRU** and **TCN** are jointly ranked first with $\Sigma_{\text{rank}} = 31$, and both lead in sequence tasks ($\Sigma_{\text{SEQ}} = 12$). Conv1D ranks second ($\Sigma_{\text{rank}} = 43$) and GRUReLU third (46). Notably, GRU, BiGRU, TCN, and Transformer share the same $\Sigma_{\text{SET}} = 19$, yet differ substantially in Σ_{SEQ} , confirming that set-task performance alone does not predict overall quality. AttentionPool and LSTM are tied at rank 14 ($\Sigma_{\text{rank}} = 99$). This pattern is visually captured in Figure 1, while Figure 2 highlights the cross-task profile differences among the top-ranked models.

For the real-world application we select three architectures from the top of the global ranking: **BiGRU** (rank 1), **TCN** (rank 1), and

GRUReLU (rank 3). The second-ranked Conv1D is skipped because it is architecturally a simplified, non-causal variant of TCN; both rely on 1D convolutions with pooling, so including both would add little architectural diversity. GRUReLU, by contrast, belongs to the recurrent family but differs substantially from BiGRU: it uses a unidirectional GRU with LayerNorm and ReLU rather than a bidirectional pass, giving it a different way of processing the input. The resulting trio—one bidirectional recurrent model, one causal convolutional model, and one unidirectional recurrent model—provides broad architectural coverage while remaining compact (under 2 000 parameters in the benchmark configuration).

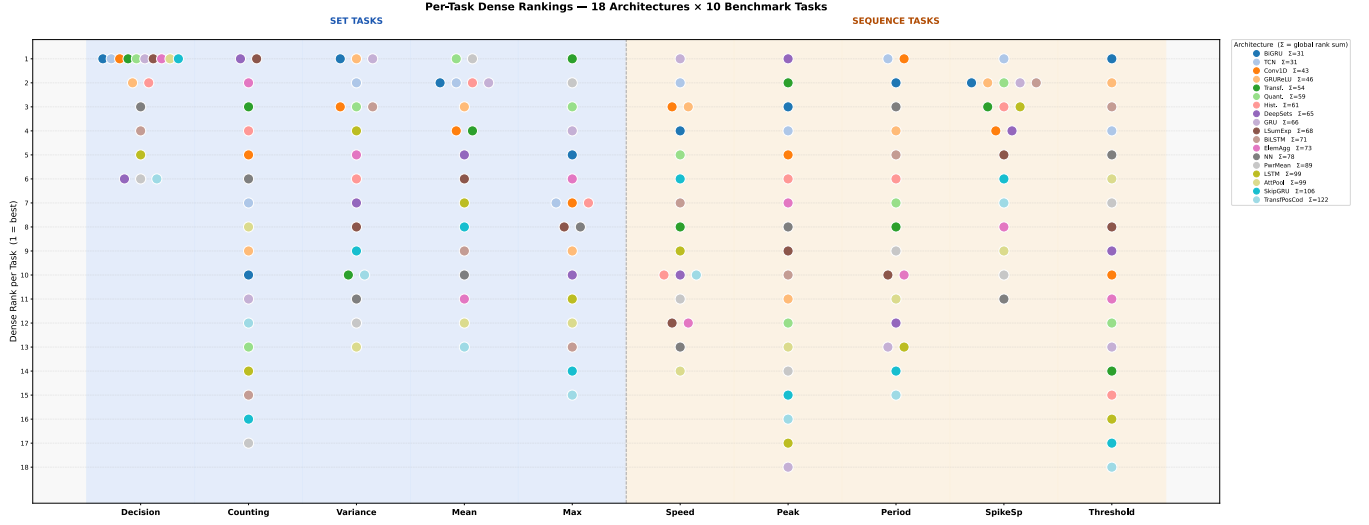


Figure 1: Per-task dense rank of all 18 architectures across the 10 benchmark tasks (1000:1000 configuration). Each point shows one architecture’s rank on a single task; legend entries are ordered by global rank sum (Σ). Blue background: permutation-invariant set tasks (Tasks 1–5); orange: order-dependent sequence tasks (Tasks 6–10).

Table 3: All architectures ranked by global rank sum (Σ_{rank} , lower is better). Σ_{SET} and Σ_{SEQ} show rank sums for set and sequence tasks, respectively.

#	Model	Params	Σ_{rank}	Σ_{SET}	Σ_{SEQ}
1	BiGRU	1,921	31	19	12
1	TCN	1,667	31	19	12
2	Conv1D	292	43	20	23
3	GRUReLU	963	46	24	22
4	Transformer	24,961	54	19	35
5	Quantile	385	59	21	38
6	Histogram	406	61	21	40
7	DeepSets	16,705	65	29	36
8	GRU	14	66	19	47
9	LogSumExp	12	68	24	44
10	BiLSTM	2,529	71	44	27
11	ElemAgg	9	73	25	48
12	NN	1,001	78	38	40
13	PowerMean	11	89	38	51
14	AttentionPool	30	99	46	53
14	LSTM	16	99	41	58
15	SkipGRU	14	106	48	58
16	TransfPosCod	29,665	122	56	66

7 Application: Gesture Speed Estimation

Having identified the top-performing architectures on abstract tasks, we now apply them to a concrete robotics-relevant problem: estimating the execution speed of dynamic arm gestures from skeletal keypoint data.

7.1 Motivation

In our prior work [3], we developed a real-time pipeline for dynamic arm gesture recognition that classifies the *type* of gesture (e.g., hand circles, hand waves, call-to-pass). That system uses OpenPose [5]

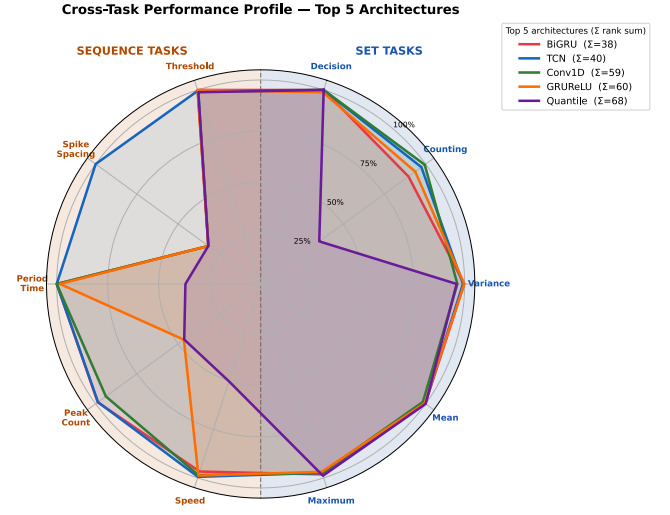


Figure 2: Cross-task performance profiles of the five architectures with the lowest global rank sum (1000:1000 configuration). Spoke values are min–max normalized across all 18 architectures (1.0 = best, 0.0 = worst). Blue sector: set tasks; orange sector: sequence tasks.

for keypoint extraction, 1×1 normalization, and a GRU-based classifier. The system was shown to be robust to variations in both viewing angle and gesture speed.

However, in many robotic applications, the *speed* at which a gesture is executed carries additional meaning beyond its type—a fast gesture may signal an urgent command, while the same gesture performed slowly indicates a routine one. Our prior pipeline cannot measure this speed; it only classifies gesture type. To address this

gap, we propose using neural networks to estimate gesture speed directly from the processed keypoint data.

7.2 Dataset

The gesture dataset consists of real recordings of eight dynamic arm gestures captured using a standard camera and processed through the OpenPose BODY-25 model:

- balraKoroz (Right Hand Circle Left) – 18 253 frames
- jobbraKoroz (Right Hand Circle Right) – 44 868 frames
- simaAllas (Standing Still) – 2 783 frames
- balMeneszt (Left Hand Wave) – 40 081 frames
- jobbMeneszt (Right Hand Wave) – 45 680 frames
- felhivasAthaladasra (Call to Pass) – 48 757 frames
- balkezJobbraKoroz (Left Hand Circle Right) – 25 720 frames
- balkezBalraKoroz (Left Hand Circle Left) – 30 568 frames

All data undergoes the same preprocessing pipeline as described in [3]. First, OpenPose BODY-25 keypoints are extracted from each frame. Positional dependence is removed by translating all keypoints so that the neck (BODY-25 keypoint 1) is at the origin. A 1×1 *normalization* is then applied: the translated keypoints are stretched horizontally and vertically so that the maximum horizontal and vertical distances between any two points become exactly 1, mapping the skeleton into a unit square while preserving its essential spatial structure. From the normalized skeleton, five joint angles are computed: the right elbow (RShoulder–RElbow–RWrist), the left elbow (LShoulder–LElbow–LWrist), the right shoulder (Neck–RShoulder–RElbow), the left shoulder (Neck–LShoulder–LElbow), and the neck-trunk angle (Nose–Neck–MidHip). Each angle is measured over the full 360° range and mapped to $[0, 1]$.

A sliding window of 50 frames is used to create input samples. As established in [3], this window length corresponds to approximately 1.67 seconds at 30 FPS, which is sufficient for recognizing a complete gesture cycle and, by extension, for estimating execution speed from the windowed signal.

7.3 Speed Ground Truth Generation

Figure 3 depicts the eight gesture types, each shown through their characteristic motion phases. For each gesture, a characteristic *starting position* (also called *null position*)—a pose that is periodically revisited during the gesture cycle—is manually identified. The distance from the starting-position skeleton is then computed frame by frame as a weighted Euclidean distance over the nine upper-body keypoints:

$$d(a, b) = \sqrt{\sum_{i=0}^8 w_i [(a_i^x - b_i^x)^2 + (a_i^y - b_i^y)^2]},$$

where $w_i = 10$ for the four distal arm joints (RElbow, RWrist, LElbow, LWrist; indices 3, 4, 6, 7) and $w_i = 1$ for all others. The elevated weight on elbows and wrists reflects their dominant role in the performed gestures; lower-body keypoints (indices 9–24) are excluded entirely.

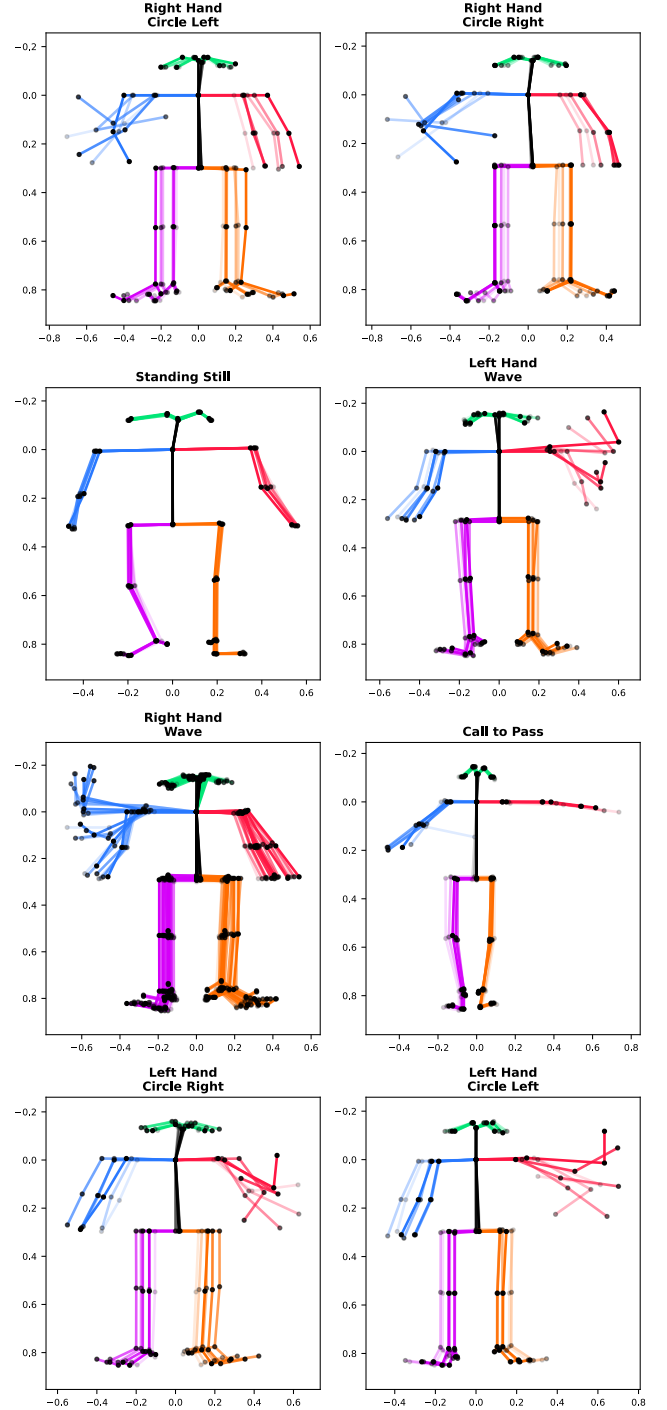


Figure 3: Reference poses for each of the eight gesture types, visualized as BODY-25 skeletal keypoints after 1×1 normalization. Blue: right arm; red: left arm; black: torso; green: head.

Computed over the entire sequence, this yields a 1D distance curve (Figure 4) whose local minima correspond to moments when

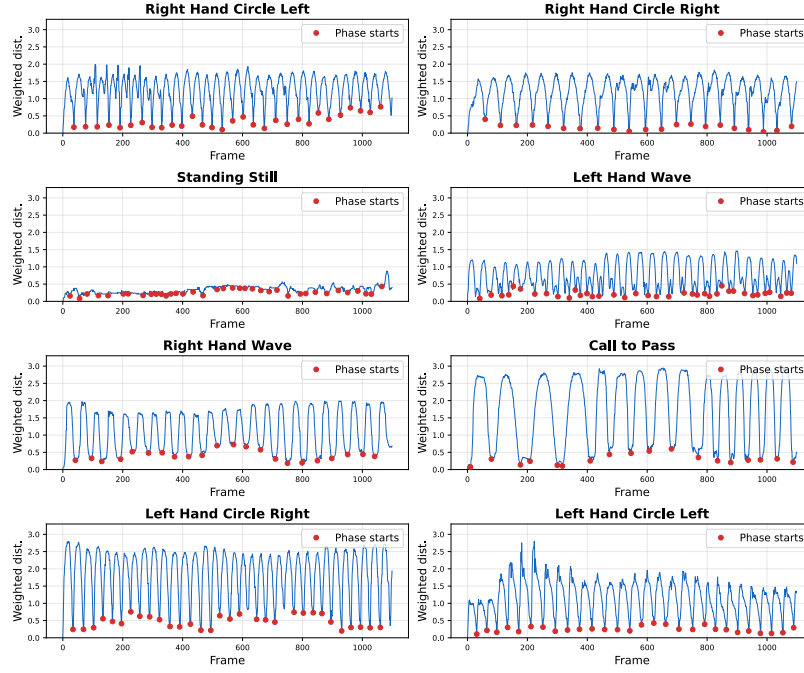


Figure 4: Weighted distance-from-reference-pose curves for example gesture sequences. Red dots mark local minima—moments at which the skeleton is closest to the reference pose. Inter-minima spacing reflects execution speed.

the gesture returns to (or passes closest to) the starting position; their temporal spacing directly encodes the execution speed. This algorithmic approach was established in [3] and serves as the labeling mechanism for all three speed interpretations described below.

7.4 Three Speed Interpretations

The speed estimation problem admits three distinct interpretations. Each corresponds to one of our benchmark tasks and captures a different—but equally valid—aspect of execution speed from the distance curve described above. This is precisely why Tasks 7, 8, and 9 were included in the benchmark: they abstract the core pattern-recognition capabilities that the application requires. Conversely, the benchmark results guide our choice of models for the application.

In all three cases, the model receives as input the 5 normalized joint angles over 50 frames. The three interpretations differ in what target value y is extracted from the local minima of the distance curve:

- (1) **Peak Count (Task 7):** The target is the number of local minima (i.e., returns to the starting position) within the 50-frame window:

$$y = |\{i : d_{i-1} > d_i < d_{i+1}\}|,$$

where d_i is the distance from the starting position at frame i . A higher count indicates faster execution. This interpretation directly counts how many complete gesture cycles fit in the window.

- (2) **Period Time (Task 8):** The target is the number of frames between the first two consecutive local minima, measuring

the duration of a single cycle:

$$y = p_2 - p_1 \quad (\text{in frames}),$$

where p_1, p_2 are the frame indices of the first two minima. A lower value indicates faster execution. Given the camera's frame rate, this can be directly converted to wall-clock time (e.g., at 30 FPS, $y = 30$ corresponds to a 1-second cycle).

- (3) **Spike Spacing (Task 9):** The target is the mean distance (in frames) between all consecutive pairs of local minima in the window:

$$y = \frac{1}{K-1} \sum_{k=1}^{K-1} (p_{k+1} - p_k),$$

where $p_1, \dots, p_K$ are the positions of the K local minima. This provides a smoothed, more robust estimate of cycle duration by averaging over all available cycles rather than relying on a single pair.

7.5 Results

Table 4 presents the results of applying the three selected architectures to the three speed interpretations on real gesture data. The dataset is split 80%/20% into training and test sets using a fixed random seed for reproducibility.

Note on parameter counts. The parameter counts in Table 4 are substantially larger than the benchmark values in Table 3 for two reasons: (i) the input dimension increases from 1 (scalar benchmark sequences) to 5 (joint-angle channels), and (ii) the hidden dimensions are enlarged to match the added complexity of real-world

Table 4: Application results on real gesture speed estimation. Bold = best per interpretation group: MAE, MSE, Std ($\downarrow$); Params ($\downarrow$ fewest); Bias ($\downarrow$ closest to zero). Bias is the signed mean error; Std is the error standard deviation.

Interpr.	Model	Params	MAE	MSE	Bias	Std.
7-Peak	BiGRU	27,649	0.198	0.097	+0.009	0.311
	GRUReLU	13,825	0.214	0.113	-0.009	0.336
	TCN	6,785	0.272	0.141	+0.065	0.370
8-Period	GRUReLU	13,057	1.320	5.942	-0.220	2.428
	BiGRU	26,113	1.499	7.157	+0.478	2.632
	TCN	6,401	2.882	16.756	+0.775	4.019
9-SpikeSp	BiGRU	27,649	2.193	11.856	+0.098	3.442
	GRUReLU	13,825	2.366	13.501	+0.008	3.674
	TCN	6,785	3.706	25.422	-0.437	5.023

multivariate data (e.g., hidden size $16 \rightarrow 64$ for GRU-based models). Together, these two factors account for the approximately $14\times$ increase in parameter count. Counts also differ slightly between interpretations for the same architecture because the output-layer normalization varies per target variable.

7.6 Interpretation of Error Metrics

The practical significance of these error values depends on the output range of each interpretation (Figure 5):

- **Peak Count:** The target range is $[0, 4]$. The best model (BiGRU) achieves $\text{MAE} = 0.198$, meaning the predicted peak count deviates from the true value by ~ 0.2 on average—a relative error of approximately **5.0%**. All three models perform well on this interpretation.
- **Period Time:** The target range is approximately $[15, 48]$ frames (range ≈ 33). The best model (GRUReLU) achieves $\text{MAE} = 1.320$, corresponding to a relative error of approximately **4.0%** with respect to the range. This means the estimated cycle duration is off by about 1.3 frames on average.
- **Spike Spacing:** The target range is approximately $[3, 30]$ frames (range ≈ 27). The best model (BiGRU) achieves $\text{MAE} = 2.193$, corresponding to a relative error of approximately **8.1%**. This interpretation is the most challenging because the model must estimate the *average* of multiple inter-cycle distances rather than a single measurement.

Overall, the **Peak Count interpretation yields the best results**, with all three models achieving practically useful accuracy. BiGRU is the strongest performer overall, achieving the best MAE on two of three interpretations, while GRUReLU offers a good accuracy-to-parameter-count trade-off (roughly half the parameters of BiGRU).

TCN, despite being jointly ranked first in the global benchmark, performs weakest on all three real-data interpretations—a reversal that is not coincidental. In the benchmark, TCN excels on scalar ($d = 1$) sequences; in the application, the input has five joint-angle channels ($d = 5$). TCN’s dilated convolutions must now share their capacity across multiple input channels within the same architectural structure, reducing their advantage for detecting 1D temporal patterns. Recurrent models (BiGRU, GRUReLU) integrate multi-channel information naturally through hidden-state updates and

are therefore more robust when input dimensionality increases. This highlights an important practical caveat: benchmark rankings on scalar sequences do not automatically guarantee proportional advantage on multivariate real-world inputs.

7.7 Error Distribution and Real-World Deployment

Beyond average-case metrics, we evaluate the full *error distribution* to assess deployment suitability: a low mean error is insufficient if individual predictions occasionally deviate wildly, as such outliers can trigger unexpected robotic behavior. We therefore report both signed mean error (Bias) and standard deviation (Std.) for each model–interpretation pair (Table 4), and inspect the full error histograms (Figure 6).

Here we distinguish two related but separate quantities: the *error distribution* (the distribution of $\hat{y} - y$, i.e. how much each prediction deviates from the true value), and the *spread of predictions around the true value* (captured by the standard deviation of that error). The former reveals systematic bias; the latter determines whether the model is consistently reliable or occasionally catastrophically wrong.

All three models exhibit near-zero bias on the Peak Count interpretation (absolute value below 0.07), confirming that systematic over- and under-prediction cancel out. More importantly, the error standard deviations remain tightly bounded: BiGRU achieves $\sigma = 0.311$ peaks on Peak Count, meaning 95% of predictions land within ± 0.62 of the true value. The histograms confirm the absence of heavy outlier tails across all interpretations—precisely the property required for reliable real-time deployment.

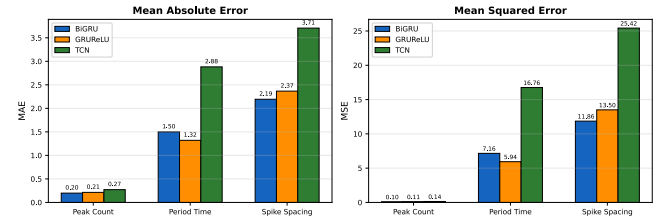


Figure 5: MAE and MSE for the three selected architectures across the three speed interpretations.

8 Conclusions and Future Work

This paper set out to answer a concrete question: *which neural network architectures are best suited for recognizing abstract temporal patterns in sequential data?* Through a systematic benchmark of eighteen architectures across ten tasks—five permutation-invariant and five order-dependent—evaluated under strictly identical conditions with more than 250 experimental configurations, we have arrived at a clear answer.

Four architectures—**BiGRU**, **TCN**, **Conv1D**, and **GRUReLU**—consistently outperform the field across both task types, occupying the top three places in the global ranking (BiGRU and TCN jointly ranked first with $\Sigma_{\text{rank}} = 31$, Conv1D ranked second with 43, and GRUReLU ranked third with 46). Crucially, all four are compact (under 2 000 parameters in the benchmark configuration), making them

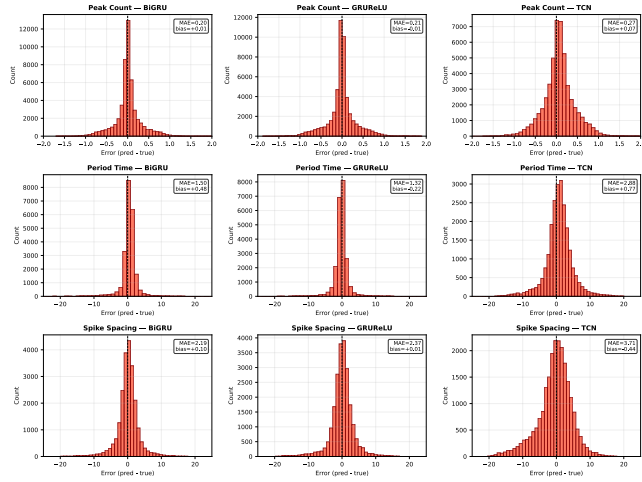


Figure 6: Prediction error distributions ($\hat{y} - y$) for each architecture-interpretation pair (rows: Peak Count, Period Time, Spike Spacing; columns: BiGRU, GRUReLU, TCN).

directly suitable for real-time deployment in resource-constrained robotic systems.

The practical value of these findings was validated through a robotics-relevant application: estimating the execution speed of dynamic arm gestures from skeletal keypoint data recorded via OpenPose. Three speed interpretations—peak count, period time, and spike spacing—were evaluated on a custom dataset of eight traffic-related gesture classes totaling 256 710 frames. The results demonstrate that neural networks can estimate gesture speed from skeletal data with high accuracy:

- Peak count: MAE = 0.198 (approximately 5% relative error) with BiGRU
- Period time: MAE = 1.320 (approximately 4% relative error) with GRUReLU
- Spike spacing: MAE = 2.193 (approximately 8% relative error) with BiGRU

These error rates are well within the tolerance required for distinguishing fast from slow gesture execution in real-time robotic control, confirming that the benchmark methodology identifies architectures that generalize to real-world data.

8.1 Future Directions

Several promising directions remain for future work:

- **End-to-end speed-aware gesture control:** Joint modeling of gesture type and speed could enable robot responses based on both *what* gesture is performed and *how fast*—e.g., fast circular motion triggers emergency stop, slow motion a gradual turn.
- **Gesture-agnostic speed estimation:** A single model that estimates speed independently of gesture identity would improve scalability by removing the requirement for known gesture types at inference time.

- **Noise robustness:** Extending the benchmark with keypoint jitter, occlusions, and natural variability would systematically evaluate architecture robustness in real-world deployment conditions.

Acknowledgments

Supported by the EKÖP-25-1-I-ELTE-872 project of the EKÖP-25 University Excellence Scholarship Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund.

References

- [1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. arXiv:1607.06450 [stat.ML]. <https://arxiv.org/abs/1607.06450>
- [2] Milán Zsolt Bagladi. 2025. Artificial Intelligence for interpreting static human arm signals. *Annales Mathematicae et Informaticae* 61 (2025), 43–54. doi:10.33039/ami.2025.10.005
- [3] Milán Zsolt Bagladi, László Gulyás, and Gergő Szalay. 2025. Fast Real-Time Pipeline for Robust Arm Gesture Recognition. In *Proceedings of the Intelligent Robotics FAIR 2025 (IntRob '25)*. ACM, 138–143. doi:10.1145/3759355.3759633
- [4] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2018. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling. *arXiv preprint arXiv:1803.01271* (2018). arXiv:1803.01271 [cs.LG]
- [5] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. 2021. OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 1 (2021), 172–186. doi:10.1109/TPAMI.2019.2929257 arXiv:1812.08008
- [6] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv:1406.1078 [cs.CL]. <https://arxiv.org/abs/1406.1078>
- [7] Jian He, Cheng Zhang, Xinlin He, and Ruihai Dong. 2020. Visual Recognition of Traffic Police Gestures with Convolutional Pose Machine and Handcrafted Features. *Neurocomputing* 390 (2020), 248–259. doi:10.1016/j.neucom.2019.07.103
- [8] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780. doi:10.1162/neco.1997.9.8.1735
- [9] Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional Recurrent Neural Networks. *IEEE Transactions on Signal Processing* 45, 11 (Nov. 1997), 2673–2681. doi:10.1109/78.650093
- [10] Thomas Stadelmayr, Youcef Hassab, Lorenzo Servadei, Avik Santra, Robert Weigel, and Fabian Lurz. 2024. Lightweight and Person-Independent Radar-Based Hand Gesture Recognition for Classification and Regression of Continuous Gestures. *IEEE Internet of Things Journal* 11, 9 (2024), 15285–15298. doi:10.1109/JIOT.2023.3347308
- [11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., 5998–6008. <https://arxiv.org/abs/1706.03762>
- [12] Thales Vieira, Romain Faugeron, Dimas Martínez Morera, and Thomas Lewiner. 2017. Online human moves recognition through discriminative key poses and speed-aware action graphs. *Machine Vision and Applications* 28, 1–2 (2017), 185–200. doi:10.1007/s00138-016-0818-y
- [13] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. 2023. Transformers in Time Series: A Survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)*. 6778–6786. doi:10.24963/ijcai.2023/759 Survey Track.
- [14] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R. Salakhutdinov, and Alexander J. Smola. 2017. Deep Sets. In *Advances in Neural Information Processing Systems*, Vol. 30. Curran Associates, Inc., 3391–3401. <https://arxiv.org/abs/1703.06114>
- [15] Mihai Zanfir, Marius Leordeanu, and Cristian Sminchisescu. 2013. The Moving Pose: An Efficient 3D Kinematics Descriptor for Low-Latency Action Recognition and Detection. In *2013 IEEE International Conference on Computer Vision*. 2752–2759. doi:10.1109/ICCV.2013.342