Computer Vision:
EqMotion Meets Assembly101

Equivariant Motion Modeling for Manipulation Tasks

Matteo Onger

September 2025

1 Introduction

This study explores the applicability of the *EqMotion* model [5] in domains beyond those previously investigated. In particular, its performance is evaluated on the prediction of hand motions during the execution of procedural tasks. The *Assembly101* [3] dataset was used to assess the model's capabilities. To enable a direct comparison with alternative approaches, EqMotion was integrated into the *4D-Hands* framework[1], facilitating a consistent evaluation setup.
In fact, the 4D-Hands framework allows multiple models to be trained and tested on exactly the same data using a shared training and test set. Furthermore, for any hyperparameter that is common to multiple models, the same value is assigned across all of them to ensure a fair and consistent comparison. This design minimizes experimental variability and strengthens the reliability of the performance evaluation. Currently, the framework includes several standard approaches used as baseline benchmarks, as well as some graph mechanics networks (GMNs) and spatio-temporal graph neural networks (STGNNs) which are equivariant, like EqMotion. A complete list is available in the framework repository.

The report is organized as follows: Section 2 provides a description of the EqMotion model, while Section 3 presents details of the dataset used. Section 4 reports the experimental results, focusing on the model's performance and its comparison with competing approaches, followed by the conclusions in Section 5.

2 Model description: EqMotion

The EqMotion model [5] was specifically designed for motion prediction tasks. In such problems, two key properties are essential: equivariance with respect to Euclidean transformations and invariance with respect to interaction relationships. Equivariance is important because the predicted motion should remain consistent even if the input is rotated, translated, or reflected, while interaction invariance ensures that the model captures the relationships between agents regardless of their absolute positions. These properties can be formally defined as follows.

Definition 2.1 (Equivariance)**.** *Let x denote an input and $f(\cdot)$ a given operation. The operation f is said to be equivariant under an Euclidean transformation if, for every transformation T, it holds that*

$$f(T(x)) = T(f(x)).$$

Definition 2.2 (Invariance)**.** *Let x denote an input and $f(\cdot)$ a given operation. The operation f is said to be invariant under an Euclidean transformation if, for every transformation T, it holds that*

$$f(x) = f(T(x)).$$

Given their importance, the EqMotion model is constructed so that all operations and transformations it performs consistently preserve these two properties. As shown in Figure 1, the model can be viewed as comprising five main components, but, first of all, let $X_i \in \mathbb{R}^{T_p \times d}$ and $Y_i \in \mathbb{R}^{T_f \times d}$ denote the past and future positions of the i-th agent in a d-dimensional space, and let $\mathbf{X} \in \mathbb{R}^{N \times T_p \times d}$ and $\mathbf{Y} \in \mathbb{R}^{N \times T_f \times d}$ denote the past and future positions of the entire system of N agents. These are the only inputs required by the model.

[1] The code is available at: https://github.com/FrancescoAgnelli3/4D_hands.

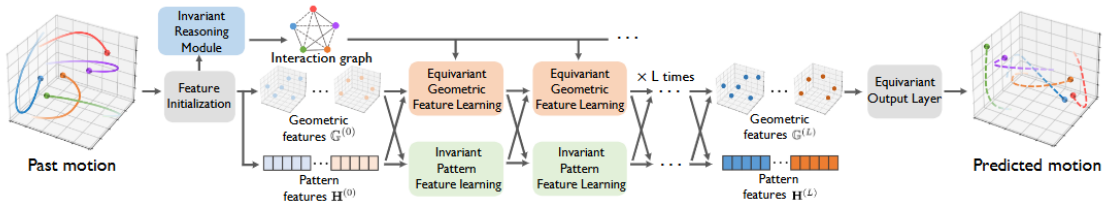

Figure 1: Schema of the architecture of the EqMotion model from the original paper [5].

The input $\mathbf{X}$ is first fed into the *feature initialization* component (Subsection 2.1), which generates two types of outputs: *equivariant geometric features* $\mathbf{G}^{(0)}$ and *invariant pattern features* $H^{(0)}$. When the interaction relationships that define the interaction graph are not known a priori, both these outputs are provided as inputs to the *invariant reasoning module* (Subsection 2.2).
The interaction categories obtained in this way are used by each of the L *equivariant geometric feature learning* layers (Subsection 2.3). At each step, each of these layers also takes as input the geometric features $\mathbf{G}^{(l-1)}$ and pattern features $H^{(l-1)}$ produced by the previous layer, and outputs updated geometric features $\mathbf{G}^{(l)}$. While the *invariant pattern feature learning* component (Subsection 2.4) operates in a similar manner, but it updates the pattern features $H^{(l)}$ without relying on interaction categories.
Finally, the geometric features $\mathbf{G}^{(L)}$ and the pattern features $H^{(L)}$, obtained at the last step, are provided as input to the *equivariant output layer* (Subsection 2.5), which produces the final motion prediction.

A detailed description of each component is presented in the following subsections. For an even more comprehensive explanation, readers are referred to the original paper and its supplementary material, particularly for the formal proofs that equivariance and invariance are preserved.

2.1 Feature initialization

The feature initialization layer is designed to obtain the initial geometric and pattern features. The geometric features $\mathbf{G}^{(0)} \in \mathbb{R}^{N\times C\times d}$ are computed according to Equation 2.1.1, where $\phi_{\text{init\_g}}$ denotes a linear transformation and $W_{\text{init\_g}} \in \mathbb{R}^{C\times T_p}$.

$$G_i^{(0)} = \phi_{init\_g}(X_i - \overline{\mathbf{X}}) + \overline{\mathbf{X}} = W_{init\_g} \cdot (X_i - \overline{\mathbf{X}}) + \overline{\mathbf{X}} \tag{2.1.1}$$

Similarly, the initial pattern features $H^{(0)} \in \mathbb{R}^{N\times D}$ are computed as the embedding of the concatenation, denoted by the operator $[\cdot;\cdot]$, of the velocity magnitude and the velocity angle, as shown by the Equation 2.1.2.

$$\begin{aligned} V_i &= \Delta X_i \in \mathbb{R}^{T_p\times d} \\ p_i^t &= \|V_i^t\|_2 \\ \theta_i^t &= \text{angle}(V_i^t, V_i^{t-1}) \\ h_i^{(0)} &= \phi_{\text{init\_h}}([p_i;\theta_i]) \in \mathbb{R}^D \end{aligned} \tag{2.1.2}$$

Here, the index i denotes the agent under consideration, C and D denote the hidden embedding dimensions, while ϕ_{init_h} represents a generic embedding function, which can be implemented for instance as a simple linear transformation or as a multilayer perceptron (MLP).

Implementation details From an implementation perspective, the operations described above are carried out directly within the `forward` method of the `EqMotion` class. Two main differences are present compared to the theoretical formulation: the velocity is provided as input rather than being computed, and the concatenation is applied after the embedding. These variations are purely implementational and do not affect the model's theoretical properties.

2.2 Invariant reasoning module

The invariant interaction module is used to infer the type of interaction between each pair of nodes when it is not known a priori - as is typically the case. A complete graph is adopted so as not to impose any a priori constraints. Consequently, one category among the K possible ones, where K

is a hyperparameter, is assigned to each pair of agents/nodes. In this way, the model learns how agents interact with one another, with the categories being computed according to Equation 2.2.1.

$$
\begin{aligned}
m'_{ij} &= \phi_{rm}([h_i^{(0)}; h_j^{(0)}; ||G_i^{(0)} - G_j^{(0)}||_{2,col}) \\
p'_i &= \sum_{j \in \mathcal{N}_i} m'_{ij} \\
h'_i &= \phi_{rh}([p'_i; h_i^{(0)}]) \\
c_{ij} &= sm\left(\frac{\phi_{rc}([h'_i; h'_j; ||G_i^{(0)} - G_j^{(0)}||_{2,col}])}{\tau}\right) \in [0,1]^K
\end{aligned}
\tag{2.2.1}
$$

[2] In the above formula, τ denotes the temperature, while each ϕ corresponds to a learnable function implemented by a MLP. Rather than relying on hard assignments, a softmax sm is applied to produce a probability distribution over the K possible classes for each pair of agents.

Implementation details This procedure is carried out within the `calc_category` method of the `EqMotion` class, with one modification: rather than only using the geometric features computed by the feature initialization layer, a velocity embedding is concatenated to them. So, only in the Equation 2.2.1, $\mathbf{G}^{(0)}$ is equal to:

$$
\mathbf{G}^{(0)} \leftarrow [\mathbf{G}^{(0)}; \phi_{rv}(\mathbf{V})] \in \mathbb{R}^{N \times 2C \times d}. \tag{2.2.2}
$$

2.3 Equivariant geometric feature learning

The equivariant geometric feature learning module consists of three main operations, described in detail below, which are executed sequentially to update the geometric features. This sequence of operations is repeated L times, where L is a hyperparameter, transforming the initial features $\mathbf{G}^{(0)}$ into the final features $\mathbf{G}^{(L)}$, which are then used by the output layer to predict motion.

Equivariant inner-agent attention An *attention mechanism* is applied to the coordinate dimensions of each agent's geometric features in order to capture and exploit temporal dependencies. Given the i-th agent's geometric feature $G_i^{(l)}$ and its pattern feature $h_i^{(l)}$, the updated geometric feature is computed as shown in Equation 2.3.1, with $\phi_{att}^{(l)}(\cdot) : \mathbb{R}^D \to \mathbb{R}^C$.

$$
G_i^{(l)} \leftarrow \phi_{att}^{(l)}(h_i^{(l)}) \cdot (G_i^{(l)} - \overline{\mathbf{G}}^{(l)}) + \overline{\mathbf{G}}^{(l)} \tag{2.3.1}
$$

Equivariant inter-agent aggregation Subsequently, once the attention mechanism has been applied, to capture spatial dependencies, *intra-agent aggregation* is performed by exploiting the previously computed interaction categories c_{ij} together with the other features. The procedure is expressed in Equation 2.3.2, where $e_{ij}^{(l)} \in \mathbb{R}^C$ are the learned aggregation weights based on the interaction graph.

$$
\begin{aligned}
e_{ij}^{(l)} &= \sum_{k=1}^{K} c_{ij,k} \cdot \phi_k^{(l)}([h_i^{(l)}; h_j^{(l)}; ||G_i^{(l)} - G_j^{(l)}||_{2,col}) \\
G_i^{(l)} &\leftarrow G_i^{(l)} + \sum_{j \in \mathcal{N}_i} e_{ij}^{(l)} \cdot (G_i^{(l)} - G_j^{(l)})
\end{aligned}
\tag{2.3.2}
$$

[2] Also in this case, each ϕ_k is implemented by a dedicated MLP, one for each category.

Equivariant non-linear function The importance of *non-linear operations* in enhancing neural network performance is well established. Therefore, as a final step following aggregation, the geometric features are transformed using the function defined in Equation 2.3.3. The key idea is to define a criterion with the invariance property to separate different conditions, and for each condition, design an equivariant transformation. It is worth noting that the terms *key* and *query*

[2] $\mathcal{N}_i$ denotes the set of neighbors of the i-th agent. However, here, as mentioned earlier, for generality a complete graph is assumed.

used here are adopted in analogy with the multi-head attention mechanism.

$$
\begin{aligned}
Q_i^{(l)} &= W_Q^{(l)} \cdot (G_i^{(l)} - \overline{\mathbf{G}}^{(l)}) \\
K_i^{(l)} &= W_K^{(l)} \cdot (G_i^{(l)} - \overline{\mathbf{G}}^{(l)}) \\
g_{i,c}^{(l+1)} &= \begin{cases} q_{i,c}^{(l)} + \overline{\mathbf{G}}^{(l)} & \text{if } \langle q_{i,c}^{(l)}, k_{i,c}^{(l)} \rangle \geq 0 \\ q_{i,c}^{(l)} - \langle q_{i,c}^{(l)}, \frac{k_{i,c}^{(l)}}{||k_{i,c}^{(l)}||_2} \rangle \cdot \frac{k_{i,c}^{(l)}}{||k_{i,c}^{(l)}||_2} + \overline{\mathbf{G}}^{(l)} & \text{otherwise} \end{cases}
\end{aligned}
\tag{2.3.3}
$$

Here, $q_{i,c}^{(l)}$ and $k_{i,c}^{(l)}$ denote respectively the c-th coordinate of the query $Q_i^{(l)}$ and the key $K_i^{(l)}$, $W_Q^{(l)}$ and $W_K^{(l)} \in \mathbb{R}^{C \times C}$, while $\langle \cdot, \cdot \rangle$ denotes the vector inner product. Finally, the geometric features for the next layer, $\mathbf{G}^{(l+1)}$, are obtained by gathering all coordinates $g_{i,c}^{(l+1)}$ across all agents.

Implementation details All these operations are implemented within the `FeatureLearning` class, which encompasses both the operations of this module and those of the invariant pattern feature learning. Of particular interest are the `inner_agent_attention`, `inter_agent_aggregation`, and `non_linear` methods, all invoked within the `forward` method of the class, which in turn is called L times by the `forward` method of the `EqMotion` class.
Once again, at the implementation level, a difference occurs: between the aggregation and the non-linear transformation, the geometric features are summed with a newly computed velocity embedding.

$$
\mathbf{G}^{(l)} \leftarrow \mathbf{G}^{(l)} + \phi_{vct}(\mathbf{V}) \tag{2.3.4}
$$

2.4 Invariant pattern feature learning

Alongside the L equivariant geometric feature learning layers, L invariant pattern feature learning layers are also present, which update the pattern features while consistently preserving their invariance. This update, in a manner similar to what is done in the invariant reasoning module, is performed at each layer as described in Equation 2.4.1.

$$
\begin{aligned}
m_{ij}^{(l)} &= \phi_m^{(l)}([h_i^{(l)}; h_j^{(l)}; ||G_i^{(l)} - G_j^{(l)}||_{2,col}) \\
p_i^{(l)} &= \sum_{j \in \mathcal{N}_i} m_{ij}^{(l)} \\
h_i^{(l+1)} &= \phi_h([p_i^{(l)}; h_i^{(l)}])
\end{aligned}
\tag{2.4.1}
$$

[2] Again, the functions ϕ_m and ϕ_h are implemented as MLPs, while $m_{ij}^{(l)}$ and $p_i^{(l)}$ can be interpreted as an edge feature and the neighbouring aggregate features in the message passing, respectively.

Implementation details These operations are implemented by the `edge_model` and `node_model` methods of the `FeatureLearning` class, and are invoked in a manner similar to the operations of the equivariant geometric feature learning component.

2.5 Equivariant output layer

After the L feature learning layers, the final geometric feature $\mathbf{G}_i^{(L)}$ and pattern feature $h_i^{(L)}$ of the i-th agent are obtained. The final prediction is computed as follow:

$$
\widehat{Y}_i = W_{out} \cdot (G_i^{(L)} - \overline{\mathbf{G}}^{(L)}) + \overline{\mathbf{G}}^{(L)} \tag{2.5.1}
$$

where $W_{out} \in \mathbb{R}^{T_f \times C}$ is a learnable weight matrix. By gathering the predictions of all agents, the final motion prediction $\hat{\mathbf{Y}}$ for the entire system is obtained.

Implementation details This last layer is directly implemented within the `forward` method of the `EqMotion` class, producing the final motion prediction.

2.6 Optional operations

In addition to the operations already described, EqMotion also allows for a series of optional transformations to be applied. For instance, the motion data inputs can be converted into the frequency domain through a Discrete Cosine Transform (DCT) as shown in Equation 2.6.1, where $W_{DCT} \in \mathbb{R}^{T_p \times T_P}$ and $W_{IDCT} \in \mathbb{R}^{T_f \times T_f}$. The Inverse DCT (IDCT) is then applied to the final prediction so as to recover the motion back in the original domain.

$$\begin{aligned} X_i &\leftarrow W_{DCT} \cdot (X_i - \overline{\mathbf{X}}) \\ \widehat{Y}_i &\leftarrow W_{IDCT} \cdot \widehat{Y}_i + \overline{\mathbf{X}} \end{aligned} \tag{2.6.1}$$

Furthermore, the pattern features can be enriched with a learnable token specific to each agent, which is directly optimized during training. Finally, residual connections can optionally be enabled within the feature learning layers, allowing the model to stabilize training and improve the flow of information across layers.
It is worth noting that these additional mechanisms are not part of the core theoretical design of EqMotion, but rather practical enhancements that can further improve its performance in specific settings.

3 Dataset description: Assembly101

The Assembly101 dataset [3] was used to evaluate EqMotion's performance against other state-of-the-art models for motion prediction. It consists of 362 unique sequences depicting people assembling and disassembling 101 toy vehicles, recorded simultaneously by 12 cameras: 8 static and 4 egocentric. Not only is this dataset larger than many of the datasets normally used in these cases, but it also presents several novel aspects that are underrepresented in existing video benchmarks: in addition to being recorded with 12 synchronized cameras, the dataset is goal-oriented, meaning that participants were not required to follow a strictly ordered recipe or script. Furthermore, participants had different skill levels, and therefore different propensities to make mistakes.

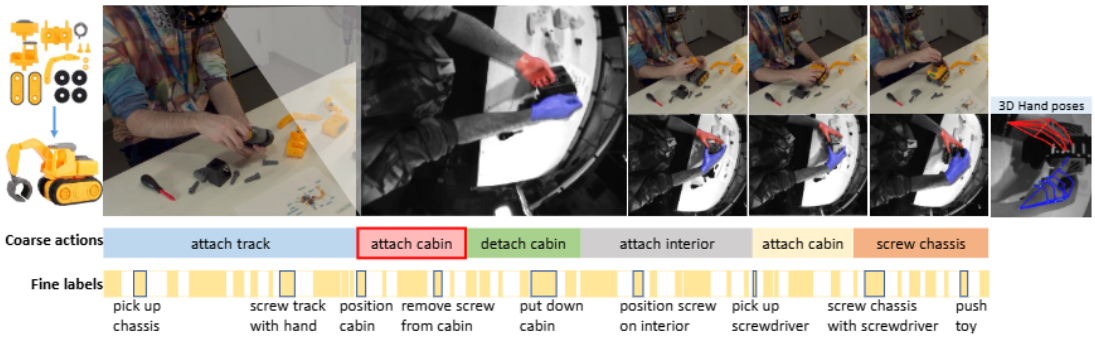

Figure 2: An example sequence from the original paper [3].

Each sequence has been carefully annotated with fine-grained and coarse actions, together with their start and end times. Each action is further categorized as correct, erroneous, or as a recovery step needed to fix a previous error. Throughout the entire sequence, the 3D positions of the hands are also continuously tracked through 42 nodes/agents (21 keypoints per hand). These detailed annotations make the dataset particularly suitable for training and evaluating models across a wide range of applications. That said, this work focused exclusively on the prediction of the 3D positions of the hands during the execution of different actions.

For further details on the dataset and its construction, we refer the reader to the original paper [3]. While in the following section, we present an analysis of the experimental results, evaluating EqMotion's performance, the influence of its most sensitive hyperparameters, and its comparison with other state-of-the-art models.

4 Experimental results

This section presents the experimental evaluation of EqMotion on the Assembly101 dataset. The Subsection 4.1 examines the model's ability to predict hand trajectories during the execution of various actions. The Subsection 4.2 investigates the effect of modifying key hyperparameters on performance, while the Subsection 4.3 benchmarks EqMotion against other approaches.
In all subsequent experiments, the dataset was limited to NumPy vectors encoding hand motion. Each vector represents, for every frame, the 3D positions of the hands through 42 keypoints (21 per hand) captured during specific fine-grained actions. The vectors were normalized and translated so that the left wrist was fixed at the origin $(0, 0, 0)$.

4.1 Action prediction

The first experiment investigates how EqMotion's performance varies when training is restricted to predicting hand movements during the execution of specific actions. All hyperparameters were kept at their default values, corresponding to a 32-dimensional latent space (for both the `hidden_nf` and the `hid_channel`) and two feature learning layers. Among the optional operations, only the Discrete Cosine Transform (DCT) was applied. The mean absolute error (MAE) was used as the loss function.

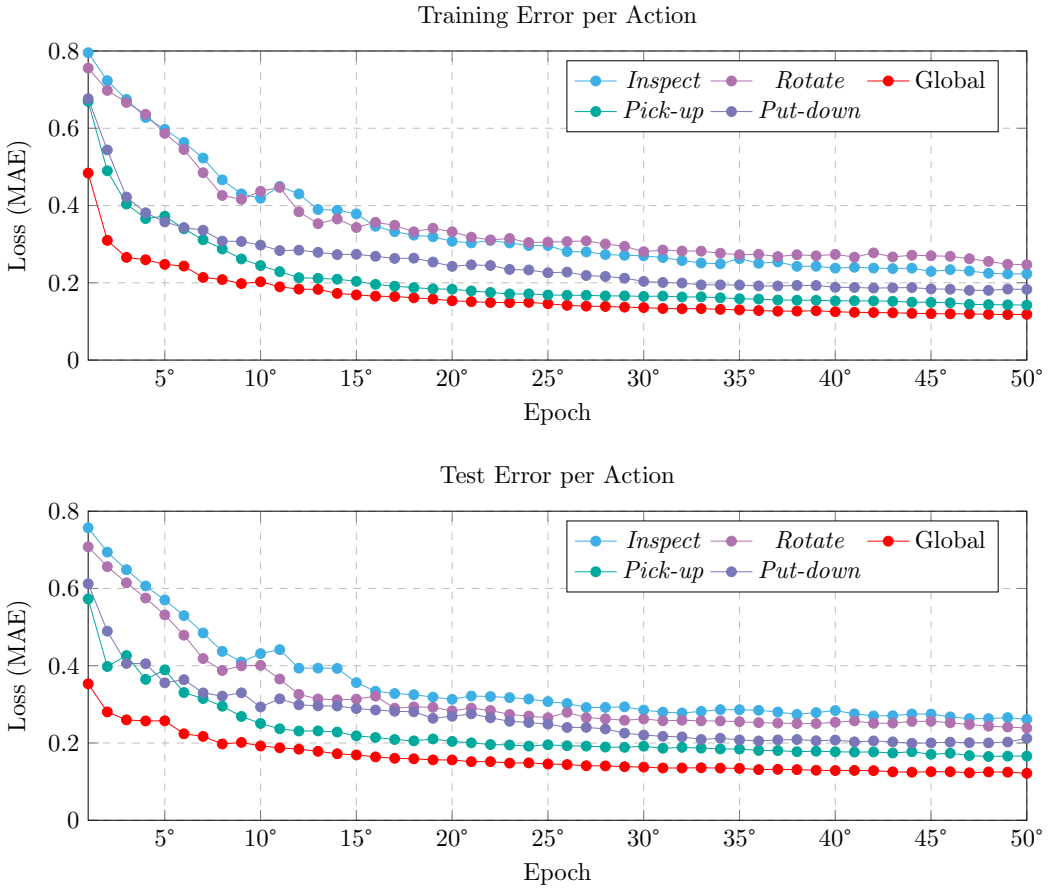

Figure 3: Training and test errors for EqMotion obtained on individual actions (*inspect*, *rotate*, *pick-up*, and *put-down*) and on the full training set (global), including all actions.

Both Table 1 and Figure 3 indicate that the model progressively improves its ability to predict hand movements as training proceeds. The best performance is achieved when the entire training set is used: although the prediction task becomes more complex due to the variability introduced by multiple actions, the larger amount of data available enables the model to handle this increased difficulty, resulting in higher accuracy.

Furthermore, it is reasonable to assume that, in addition to dataset size, performance is also influenced by the intrinsic variability of the movements associated with a given action. For instance, inspecting an object involves movements that depend strongly on the characteristics of the object

itself, and therefore exhibit greater variability than the relatively consistent motions required to pick-up an object. This variability appears to be a significant factor affecting the accuracy achieved. In fact, the final test error for the *inspect* action is almost 0.1 higher than that obtained for the *pick-up* action.

Training & Test Error per Action										
Epoch	*Inspect*		*Rotate*		Global		*Pick-up*		*Put-down*	
	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$
1°	0.80	0.76	0.76	0.71	0.48	0.35	0.67	0.57	0.68	0.61
5°	0.60	0.57	0.59	0.53	0.25	0.26	0.37	0.39	0.36	0.36
10°	0.42	0.43	0.44	0.49	0.20	0.19	0.24	0.25	0.30	0.29
25°	0.30	0.31	0.31	0.27	0.15	0.15	0.17	0.20	0.23	0.25
35°	0.25	0.28	0.27	0.25	0.13	0.13	0.16	0.18	0.19	0.21
50°	0.22	0.26	0.25	0.24	0.12	0.12	0.14	0.17	0.18	0.21
Size	163		148		2216		514		631	

Table 1: Training and test errors of EqMotion for different actions (*inspect*, *rotate*, *pick-up*, and *put-down*) and for the global setting, reported at selected epochs, with the number of training samples shown in the last row.

4.2 Impact of hyperparameters

The first hyperparameter analyzed is L, namely the number of feature learning layers. This parameter directly determines the number of trainable weights in the model and, consequently, the level of complexity it can handle. In these tests, all other hyperparameters were fixed at their default values, as in the previous experiment. The entire training set, comprising all actions, was used, and the mean absolute error (MAE) was adopted as the loss function.
The results are summarized in Table 2 and illustrated in Figure 4.

Training & Test Error by Number of Feature Learning Layers			
N° of Layers	N° of parameters	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$
2	69,188	0.118	0.124
3	91,044	0.115	0.122
4	112,900	0.114	0.120
5	134,756	0.113	0.119
6	156,612	0.107	0.116
7	178,468	0.107	0.115

Table 2: Training and test errors obtained by varying the number of feature learning layers L. The corresponding number of trainable parameters is also reported.

As the number of layers increases, the number of parameters grows substantially: raising the number of layers from 2 to 7 results in a 258% increase in model parameters. At the same time, however, the improvement in performance remains limited. This indicates a fundamental difficulty for the model in further enhancing its predictions, at least given the amount of data available.

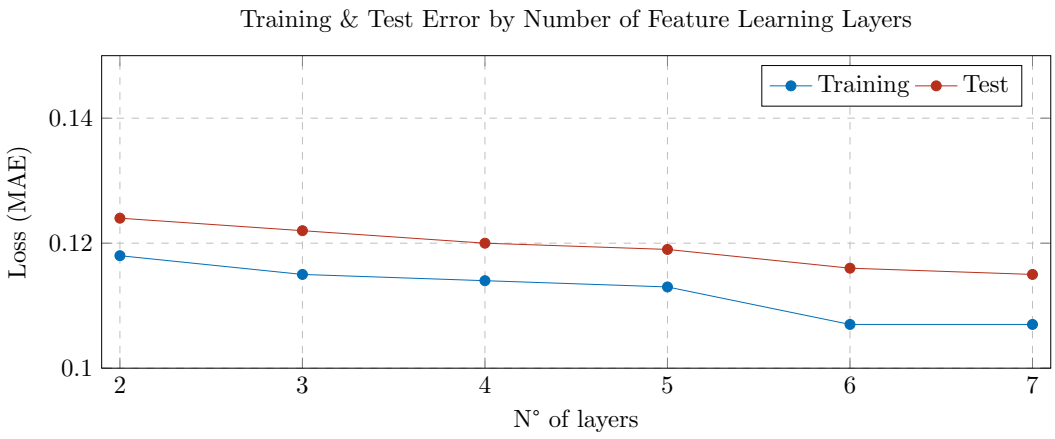

Figure 4: Training and test errors as a function of the number of feature learning layers L.

Subsequently, the impact of the optional operations, described in the Subsection 2.6, on the model's performance was examined.

Training & Test Error w/ Optional Operations								
Epoch	Default		w/ DCT		w/ Residuals		w/ Agent Token	
	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$
1°	0.21	0.21	0.44	0.31	0.22	0.22	0.20	0.20
5°	0.14	0.14	0.20	0.19	0.15	0.15	0.14	0.14
10°	0.13	0.13	0.16	0.16	0.13	0.13	0.12	0.12
35°	0.11	0.11	0.12	0.12	0.11	0.11	0.11	0.11
50°	0.10	0.10	0.11	0.12	0.11	0.11	0.10	0.10

Table 3: Training and test errors obtained with the default configuration and with the application of optional operations (DCT, residual connections, and agent tokens), reported at selected epochs.

As shown in Table 3, after 50 epochs the loss values remain broadly similar, with a slight improvement observed when DCT is excluded and either residual connections or agent tokens are applied. Even when these operations are combined, the loss does not fall below 0.10 in either the training or testing phases. At most, as illustrated in Figure 5, the learning speed changes slightly, but this does not affect the final performance.

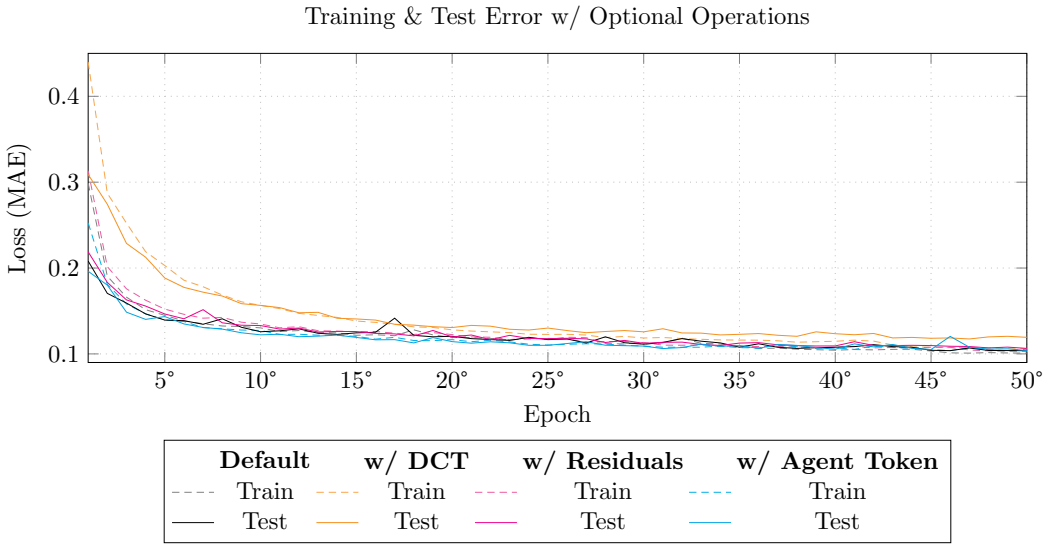

Figure 5: Training and test errors obtained with the default configuration and with the application of optional operations.

Regarding the remaining two main hyperparameters, that are the number of categories (K, referred to as `num_categories` in the code) and the size of the latent spaces (C and D, referred to as `hid_channel` and `hidden_nf` in the code), the situation differs.

Figure 6: Training and test errors as a function of the number of categories K.

Both influence the total number of parameters, with the latent spaces size having a stronger effect than the number of categories. However, as shown in Figure 6, the number of categories alone does not significantly affect the loss after the standard 50 epochs, whereas the hidden dimensions can have a stronger impact.

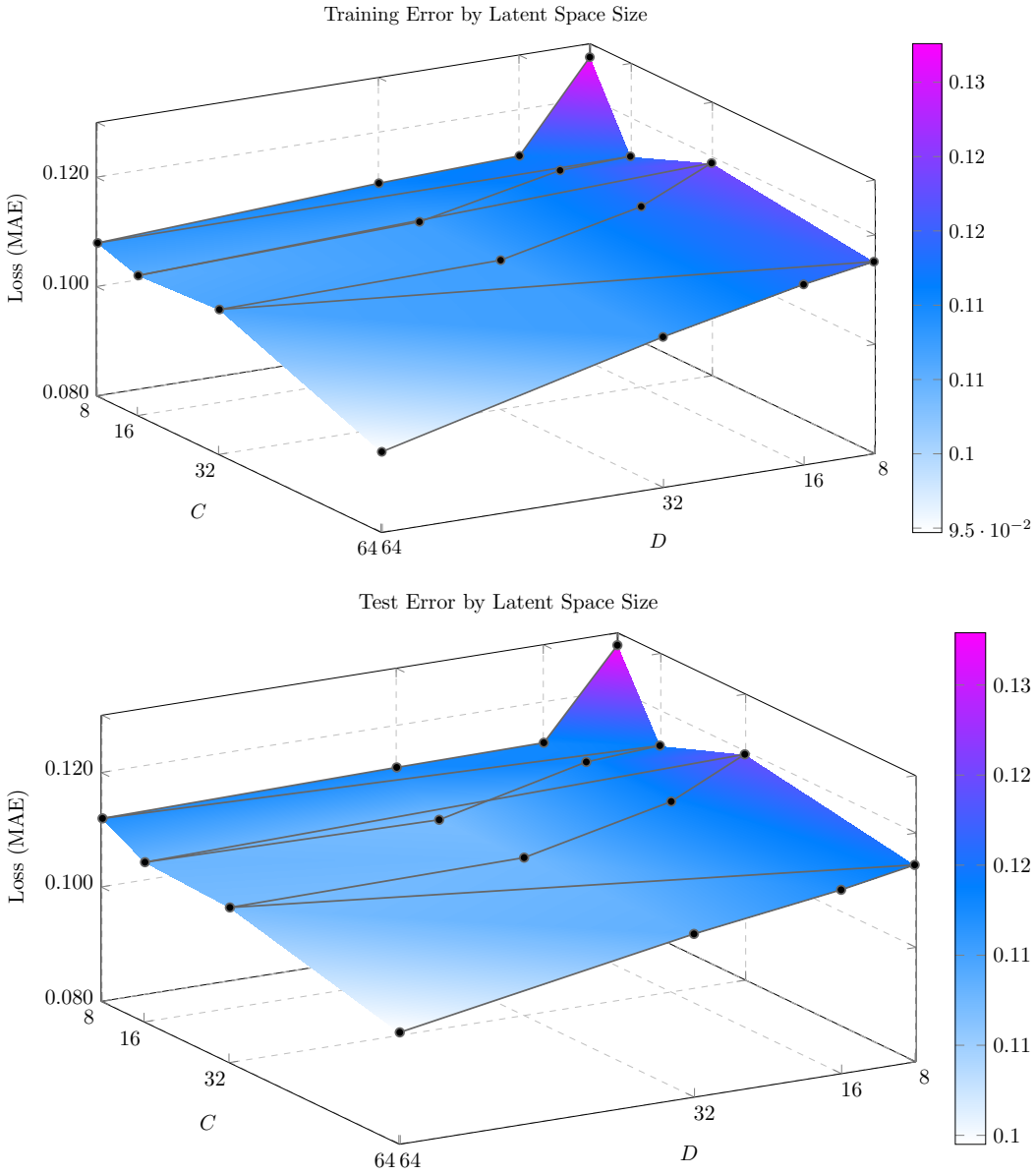

Figure 7: Training and test error as a function of the hidden dimension of the geometric features (C) and of the pattern features (D).

As shown in Figure 7, enlarging the latent space of both the geometric features (C) and the pattern features (D) generally leads to a reduction in the final loss. In fact, when both dimensions are set above 64, the model is able to break the 0.10 loss threshold. This improvement, however, comes at a cost. Increasing the size of the latent spaces inevitably raises the number of trainable parameters and, consequently, the overall complexity of the model. The effect is particularly evident when the hidden dimension of the geometric features is increased. Table 4 illustrates this trade-off clearly: a modest improvement in loss, on the order of only 0.03, corresponds to a growth in the number of parameters from just over 6,000 to more than 257,000. Naturally, such an expansion also translates into greater computational demands, which, given the available hardware, results in significantly longer training times.

It is also worth noting that, across all the experiments carried out so far, the training error has consistently remained relatively low, with no substantial differences compared to the test error. This suggests that the model suffers from neither severe underfitting nor overfitting.

Training & Test Error by Latent Space Size					
		D			
		8	16	32	64
C	8	0.128/0.128 (6K)	0.112/0.113 (11K)	0.111/0.112 (24K)	0.108/0.112 (66K)
	16	0.113/0.114 (14K)	0.112/0.113 (20K)	0.107/0.107 (36K)	0.106/0.108 (83K)
	32	0.119/0.119 (40k)	0.113/0.113 (66K)	0.107/0.108 (69k)	0.106/0.107 (127K)
	64	0.115/0.114 (134K)	0.113/0.112 (148K)	0.107/0.108(178K)	0.095/0.098 (257K)

Table 4: Training and test error by latent space size. Rows correspond to the hidden dimension of geometric features (C) and columns to the hidden dimension of pattern features (D). Each cell reports training/test error, with the total number of parameters in parentheses.

4.3 Benchmarking

In this final subsection before the conclusions, the EqMotion model is compared with a set of reference models commonly adopted for similar tasks. The comparison includes three groups of algorithms: (i) *baseline models*, which serve as simple benchmarks; (ii) *equivariant models* [2, 4], which, like EqMotion, incorporate properties of equivariance in their design; and (iii) *Graph Mechanics Networks (GMN)* [1], which are also equivariant and represent another established approach in this domain.[3]

As mentioned at the beginning, in these experiments only the default hyperparameters of each model were adopted, ensuring that the same values were used whenever hyperparameters were shared across different models. With regard to the optional operations of EqMotion, only the residual connections and the agent tokens were enabled.

Training & Test Error												
	Model	Epoch										Size
		1°		5°		10°		25°		50°		
		$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	$\mathcal{L}_{\text{train}}$	$\mathcal{L}_{\text{test}}$	
	EqMotion	0.244	0.188	0.142	0.140	0.126	0.123	0.113	0.114	0.102	0.103	74K
Baseline	Constant	0.127	0.124	0.126	0.124	0.127	0.124	0.127	0.124	0.127	0.124	—
Baseline	GWNET	0.345	0.306	0.301	0.291	0.297	0.284	0.142	0.135	0.127	0.126	334k
Baseline	Temporal	0.502	0.361	0.166	0.165	0.146	0.148	0.133	0.131	0.116	0.116	15K
Baseline	GCN	0.525	0.382	0.171	0.162	0.170	0.161	0.136	0.136	0.128	0.128	17K
Baseline	LSTM	0.525	0.348	0.231	0.212	0.172	0.170	0.140	0.139	0.129	0.129	558k
Baseline	GAT	0.484	0.381	0.171	0.159	0.150	0.169	0.130	0.126	0.122	0.118	17K
Equivariant	ENN	0.088	0.081	0.080	0.076	0.078	0.073	0.079	0.074	0.075	0.072	3K
Equivariant	ETNN	0.094	0.081	0.080	0.076	0.078	0.073	0.075	0.072	0.074	0.070	17K
Equivariant	EGNN	0.241	0.206	0.217	0.201	0.210	0.193	0.186	0.171	0.151	0.140	17K
Equivariant	ESTGCN	0.166	0.113	0.090	0.085	0.084	0.077	0.078	0.074	0.076	0.072	44K
GMN	ATT	0.091	0.079	0.079	0.075	0.078	0.073	0.075	0.071	0.074	0.071	42K
GMN	LSTM	0.300	0.214	0.107	0.098	0.091	0.086	0.083	0.078	0.080	0.076	33K
GMN	GMN	0.256	0.181	0.106	0.098	0.091	0.087	0.084	0.081	0.080	0.077	27K
GMN	LSTM INV	0.189	0.160	0.112	0.104	0.098	0.093	0.087	0.008	0.082	0.077	33K

Table 5: Comparison of training and test errors across different models at selected epochs. The table includes baseline models, equivariant architectures, and GMN-based approaches, with EqMotion reported separately. The number of parameters is also provided for each model.

Firstly, by looking at the first group of models in Table 5, it can be observed that EqMotion consistently outperforms all baseline models. This holds true even when compared to architectures with a considerably larger number of parameters, such as LSTM and GWNET. In fact, given the limited amount of available data (even when all actions are considered), models that do not exploit equivariance struggle to achieve performance superior to that of a simple constant predictor, which is precisely what is observed in these cases.

The situation changes when looking at the remaining models, namely the other equivariant and GMN-based architectures, which also exploit equivariance in their design. In comparison with these models, EqMotion shows worse performance in several cases, although often only slightly, and it only manages to achieve a lower loss only compared to EGNN [2].

[3]For more information about other models, please refer to the 4D-Hands documentation.

Interestingly, some of the most effective models reach strong performance while using a substantially smaller number of parameters than EqMotion. For example, the ENN model achieves a loss of 0.072 with just 3,000 parameters, a level of accuracy that EqMotion has not been able to reach, even when increasing the size of the latent space or the number of feature learning layers.

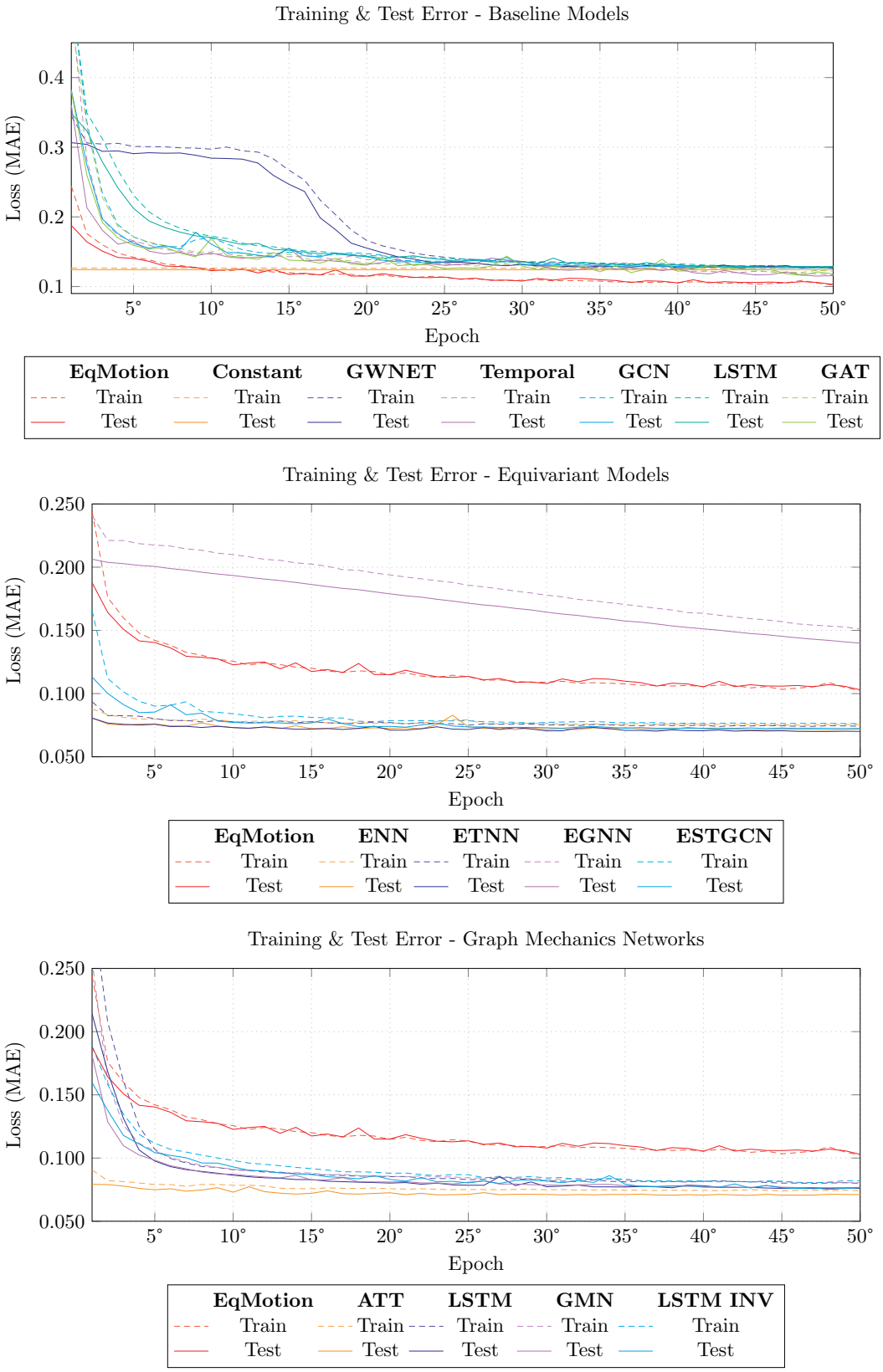

Figure 8: Training and test errors for the models considered in Table 5, grouped by category: baseline models (top), equivariant architectures (middle), and GMN-based models (bottom).

Although all equivariant and GMN-based models are often grounded in similar principles, such as leveraging invariance or equivariance and effectively capturing space–time dependencies, each of them translates these principles into practice in different ways. And, as the architectures grow in complexity, it becomes increasingly difficult to understand which specific component or design choice is responsible for their superior performance in a given task, exactly like in this case.

5 Concluding remarks

In conclusion, this work investigated the performance of the EqMotion model in predicting 3D hand movements during procedural tasks, using the Assembly101 dataset. The experiments focused on different aspects of the model, including the influence of hyperparameters, and a comparison with other state-of-the-art models.

The results show that EqMotion is effective when trained on the full dataset, with performance influenced not only by the amount of data but also by the variability of the movements associated with each action, with more complex actions leading to higher prediction errors.

The study of hyperparameters highlighted that increasing the number of feature learning layers and the size of the latent space can improve accuracy, but at the cost of a substantial increase in model parameters and longer training times. While optional operations showed only minor effects on the final loss.

Finally, when compared with other models, EqMotion outperforms baseline architectures by a significant margin, thanks to its equivariant design. However, other equivariant and GMN-based models achieve similar or better performance, often with a lower number of parameters.

Obviously, this work represents only a preliminary study. Further theoretical and experimental analyses will be needed to gain a deeper understanding of the strengths and limitations of these models, with the ultimate goal of improving their efficiency and robustness.

References

[1] Wenbing Huang et al. “Equivariant graph mechanics networks with constraints”. In: *arXiv preprint arXiv:2203.06442* (2022).

[2] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. “E (n) equivariant graph neural networks”. In: *International conference on machine learning*. PMLR. 2021, pp. 9323–9332.

[3] Fadime Sener et al. “Assembly101: A large-scale multi-view video dataset for understanding procedural activities”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 21096–21106.

[4] Liming Wu et al. “Equivariant spatio-temporal attentive graph networks to simulate physical dynamics”. In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 45360–45380.

[5] Chenxin Xu et al. “EqMotion: Equivariant Multi-Agent Motion Prediction With Invariant Interaction Reasoning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2023, pp. 1410–1420.