Virtual Sensing through Transformers and Selective State Spaces

Loucas Papalazarou

Master of Science
School of Informatics
University of Edinburgh
2024

Abstract

This dissertation investigates the application of modern machine learning techniques, specifically Transformers and Selective State Spaces (Mamba), to replicate and replace physical sensors in robotic systems through virtual sensing. The primary objective is to develop models capable of inferring sensoric outputs from a subset of sensors, thereby reducing the number of physical sensors needed and lowering associated costs. The study involves setting up a simulation environment using the Franka Emika Panda robot to perform a defined task, generating data on various measurements including positions, orientations, force sensor outputs, and images. Multiple experiments were conducted to evaluate the effectiveness of the chosen architectures, with a focus on understanding the impact of model complexity and context size on performance. Despite facing challenges related to model complexity, data handling, and computational resources, the research provides valuable insights into the feasibility of virtual sensing. The results indicate that the models could only predict the average sensor values and struggled to capture detailed sensor data nuances, highlighting the need for more advanced models and better training techniques. This work lays the foundation for future exploration in reducing robotic sensor costs through virtual sensing, with recommendations for employing more complex models and optimizing data handling strategies.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Loucas Papalazarou*)

Acknowledgements

Firstly, I would like to express my deepest gratitude to my supervisor, Dr. Michael Mistry, for his unwavering support, constructive ideas, and invaluable suggestions. I am also thankful to my friends for making this journey enjoyable and lighthearted. Most importantly, I extend my profound thanks to my family for providing me with all the opportunities I could ever hope for.

Table of Contents

Chapter 1

Introduction

1.1 Motivation

In recent years, the field of robotics has experienced significant advancements, resulting in increased accessibility and application of robotic systems [25, 38]. These systems already play a crucial role in various industries, notably in the manufacturing of everyday products like automobiles. Consequently, it is reasonable to anticipate that robots will soon become integrated into our daily lives as well.

These developments underscore the growing demand for robotic sensors, which are essential for enabling robots to perceive and interpret their environment. While it is generally accepted that incorporating more sensors can enhance a robotic system's perception capabilities, this also leads to increased costs. For the mass production of robots to be economically viable, cost-effective solutions are imperative.

This dissertation aims to investigate whether modern machine learning methods can be utilized to replicate and replace some of the sensors in a robotic system through the rest of the sensors in the broader context, a concept referred to as *virtual sensing*.

1.2 Objective and Contribution

The primary objective of this dissertation is to develop a machine learning model capable of inferring sensor outputs from a subset of sensors in a robotic system. This involves collecting various measurements, such as positions, orientations, outputs from force sensors, and images from cameras, while a robot is performing a specific task. The collected data will be used to train and evaluate modern machine learning architectures to determine their effectiveness in predicting sensor outputs.

This exploratory project aims to conduct a feasibility study and investigate which approaches work best. While many machine learning architectures were considered, the focus was primarily on the two most promising ones: the Transformer [43] and Selective State Spaces, aka Mamba [12]. The other architectures will be only briefly mentioned.

This dissertation will contribute to both academia and industry by advancing the understanding and application of virtual sensing in robotic systems. For academia, it provides an investigation into the feasibility and effectiveness of using machine learning methods, specifically Transformers and Selective State Spaces, to replicate or replace physical sensors. This research adds to the existing knowledge on robotic perception and offers a basis for future studies on cost-effective sensor solutions. For the industry, the findings have practical implications for reducing the costs of robotic systems. By demonstrating how machine learning can predict sensor outputs, this work can lead to more affordable and efficient robots, potentially speeding up their integration into various sectors.

1.3 Methodology

The work of this dissertation was conducted in an agile fashion, involving continuous testing, reviewing of literature, and adapting our approach based on new insights and findings. In this section, we describe the process we followed.

Our first step was to conduct a comprehensive literature review on how other researchers have solved the *virtual sensing* problem. The insight gained allowed us to identify gaps in the literature and formally define the task we aimed to tackle, along with a rough idea of how to approach it.

With a theoretical framework in place, we set up a simulation environment to start collecting data on a robot performing the chosen task. Detailed information can be found in Chapter 3.

Our initial approach was to use generative audio models such as WaveNet [30], WaveRNN [17], and WaveGlow [34]. The rationale behind this choice was the ability of these models to capture relationships in sequences of data. Although this approach was promising in theory, adapting audio models to our specific task proved challenging and not worth the return on time investment. For this reason, we decided to pivot to the Transformer architecture [43] due to its proven track record [1] in time-sequence tasks such as Natural Language Processing. During this phase, we also explored Mamba [12],

a recently developed and promising machine learning architecture with traits similar to the Transformer. Mamba was created to address challenges in sequence modeling and long-range dependencies in data, making it a relevant addition to our research. Additionally, as we needed a baseline to compare to, we chose the simplest counterpart of our timeseries models, the simple Recurrent Neural Network (RNN) [28].

Having established the final model architectures for our study, we designed and conducted a series of experiments to obtain the insights we sought. Following the execution of these experiments, we interpreted the results and formulated our final conclusions.

1.4 Document Structure

Chapter 2 provides a comprehensive literature review, examining how this problem has been approached by other researchers and the methods they used. It then explains the key concepts of time series forecasting and justifies why our definition of the problem falls into this category. Additionally, it introduces the machine learning architectures explored in this dissertation, explaining the rationale for selecting them to address our problem.

Chapter 3 defines the problem formally and outlines our technical approach. It details the implementation specifics, including data characteristics such as size, collection methods, and format. Additionally, it describes our experimental environment, explains the experiments conducted, and presents our results, which are briefly analyzed.

Finally, Chapter 4 provides a summary of the work done for this dissertation, discussing the implications of the results and whether the objectives we set out to achieve were met. It concludes with suggestions for improvements and directions for future work.

Chapter 2

Background

2.1 Literature Review

Limited progress has been made in the field of machine learning (ML)-assisted robotic sensing, with only a few notable studies. These works mainly focus on using machine learning and latent representations of robotic sensor data in a multi-modal approach to predict outcomes. In particular, [4, 3] employed self-supervised learning and convolutional neural networks to effectively integrate sensor data, enhancing a robot's ability to predict grasp outcomes on objects and thereby improving its performance. These findings are further supported by [23, 24], who trained machine learning models to develop multi-modal representations for contact-rich tasks using a combination of self-supervised learning and reinforcement learning, achieving moderate success.

Additionally, [22] conducted research closely related to this paper's project, exploring how data from various sensors can be combined to provide more reliable information. They introduced the Crossmodal Compensation Model (CCM), which identifies and compensates for faulty sensors. By leveraging data from functional sensors, CCM can still learn useful information, facilitating the development of effective manipulation strategies. This indicates that it is possible to estimate a sensor's reading based on others, even if some sensors are malfunctioning, suggesting that the proposed project in this paper has a solid foundation for success.

2.2 Time Series Forecasting

Most notably, none of the relevant and recent literature has modeled this problem as a time series problem. This is the main basis for our approach, but let us first articulate

what time series forecasting is.

Time series forecasting involves predicting future values based on previously observed values. It is commonly used in various domains such as finance, weather prediction, and inventory management. By treating the data as a sequence of observations indexed over time, time series analysis can capture temporal dependencies and trends that traditional methods might overlook. Unlike linear regression [29], which typically handles a single feature, time series forecasting allows for multiple features, hence accommodating higher dimensions. This approach can enhance predictive accuracy and provide a more nuanced understanding of the underlying dynamics.

In the context of ML-assisted robotic sensing, framing the problem as a time series allows us to leverage temporal patterns in sensor data, which can improve the prediction of outcomes and detection of anomalies.

The first well-known and arguably simplest architecture to tackle the time series problem with its design was the recurrent neural network (RNN) [28]. At the time, RNNs represented a significant achievement and were utilized by notable applications such as Google Translate [2]. However, due to their specific architecture, RNNs are very slow to train, which was their main drawback. Since then, numerous architectures have been proposed, such as Long Short-Term Memory (LSTM) networks [15], Gated Recurrent Units (GRUs) [5], Temporal Convolutional Networks (TCNs) [21] and others, each with their own relative success.

While numerous prominent architectures exist, our work focuses on exploring two specific ones: the widely adopted Transformer architecture [43] and the emerging Mamba architecture [12]. Mamba shares a similar architectural philosophy with the Transformer and exhibits promising potential. Detailed analyses and discussions of these architectures is provided in the following sections.

2.2.1 Transformers

The Transformer architecture, introduced by Vaswani et al. in 2017 [43], revolutionized natural language processing (NLP) and sequential data modeling. Unlike traditional recurrent neural networks (RNNs) and convolutional neural networks (CNNs), Transformers rely entirely on self-attention mechanisms, eliminating the sequential nature of RNNs and the locality assumptions of CNNs. The architecture of the Transformer is depicted in Figure 2.1.

What makes Transformers particularly effective is their ability to capture long-

range dependencies in sequential data. This is achieved through multi-head attention mechanisms, where the model attends to different positions of the input sequence simultaneously. Multi-head attention allows Transformers to weigh the importance of different words or elements in the sequence based on their contextual relevance, leading to superior performance in tasks requiring understanding of global dependencies and relationships.

In our use case, where understanding long relationships in sequential data is crucial, multi-head attention is advantageous. It enables the model to efficiently process and relate elements across the entire sequence, making Transformers well-suited for tasks such as time series forecasting, where capturing complex temporal patterns and dependencies is paramount. The way in which we use the Transformer will become clearer in Chapter 3 when we finally introduce the problem formulation.

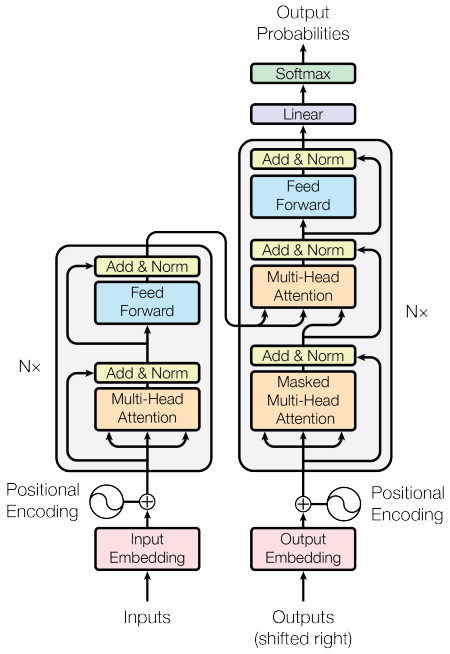


Figure 2.1: The Transformer - model architecture from [43]

2.2.2 Selective State Spaces (Mamba)

Selective State Spaces, also known as Mamba [12], represents a novel approach inspired by Transformers' design philosophy. Its architecture emphasizes selective attention mechanisms akin to multi-head attention in Transformers, enabling efficient processing and complex relationship capture within input sequences.

Mamba addresses computational inefficiencies on long sequences observed in archi-

tectures like Transformers. Unlike linear attention or gated convolution models, which have limitations in content-based reasoning, Mamba introduces selective state space models (SSMs). These SSMs dynamically adapt parameters based on input content, facilitating selective propagation or forgetting of information along the sequence length dimension.

Recent advancements show that state-space models (SSMs) like Mamba can achieve comparable or superior performance to Transformers at smaller to medium scales [6]. This highlights a close relationship between these model families, supported by theoretical frameworks connecting SSMs with various attention mechanisms.

Like Transformers, Mamba moves away from traditional sequential processing paradigms such as RNNs, offering fast inference and linear scaling in sequence length. This scalability makes Mamba effective across modalities like language, audio, and genomics, demonstrating state-of-the-art performance in tasks such as language modeling and downstream evaluations, making this architecture conceptually capable in solving our problem.

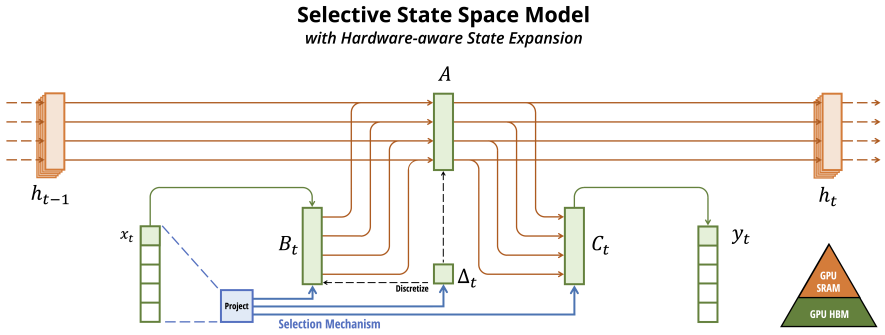


Figure 2.2: Illustration of how Structured State Space Models (SSMs) manage input data. Each segment of the input x is processed independently to produce an output y using a higher-dimensional state h. Unlike previous approaches, Mamba incorporates a dynamic selection mechanism that adjusts based on the input, necessitating efficient GPU memory management. Image from [12]

Chapter 3

Experimental Evaluation

In this chapter, we present the methodology and results of our experimental evaluation. We will discuss the simulation environment, data collection process, problem formulation, model implementation, and training strategies. This comprehensive evaluation aims to demonstrate the effectiveness and efficiency of the models and approaches used in our study.

Before delving into the technical details of this work, it is crucial to acknowledge that all decisions we made, including parameters, types of experiments, configurations, and other factors, were informed by empirical knowledge and our best judgment, given the constraints of our current understanding and the time limitations. We have endeavored to be as thorough as possible in explaining the rationale behind every decision to ensure clarity and transparency in our methodology.

3.1 Simulation Environment

Given the scarcity of real robotic instruments and the tight timeframe that constrained this dissertation, our experiments were conducted using data generated in a simulation. In order for us to articulate what is the exact task we wanted to do, we must first describe the simulation environment.

We used Isaac Gym [26], a framework for creating simulations developed by NVIDIA. Within this simulation, we have a virtual representation of the Franka Emika Panda research robot [11], along with an agent programmed to control its actions.

The agent controlling Franka has been trained using a reinforcement learning algorithm, the specifics of which are beyond the scope of this project. The primary objective of the agent is to execute a defined task, facilitating the generation of simulated

episodes and the collection of data. For us to be able to infer the output of one of the robot's sensors, we needed the robot to perform a simple yet general enough task for us to generate data from. The task we chose involves pushing a red cube onto a green cube, as depicted in Figure 3.1.

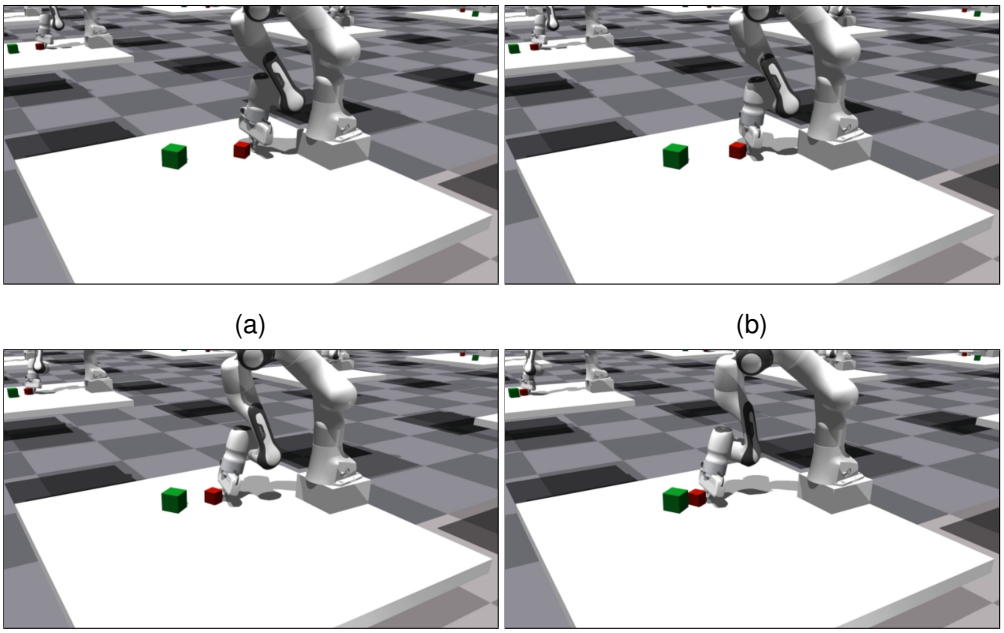


Figure 3.1: Selected frames illustrating different stages of the robot successfully completing the Cube Push Task ($a \rightarrow b \rightarrow c \rightarrow d$)

3.2 Data

In this section, we will delve into the methodologies of data collection, the formats in which the data is structured, and a detailed examination of the various data points involved. Due to the slight complexity of our dataset, we will explain it using a top-down approach. This means starting at the folder level and progressively explaining down to the individual tensor dimensions.

Our dataset is organized within a single folder, containing 875 data files. Each file is a `.pt` file with a size of `2.7GB`, culminating in a total dataset size of `2.36TB`. Despite the abundance of data available due to the ease of its generation, we opted to utilize only a subset of the data to avoid the inefficiencies associated with processing the entire dataset. The specific subset used was dynamically determined through the

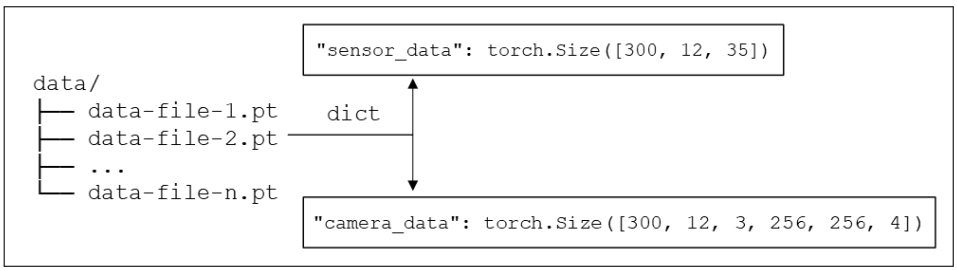


Figure 3.2: Data Structure

`data_portion` parameter, which ranges from 0 to 1 and indicates the percentage of the data employed.

Figure 3.2 provides an illustration of the data. Each of those data files contains a Python dictionary with two keys, `"sensor_data"` and `"camera_data"` with their respective values being tensors of shape `[300, 12, 35]` and `[300, 12, 3, 256, 256, 4]`. We chose to separate the sensor and camera data due to their different dimensions at the feature level, however they still refer to the same simulation runs through the first and second dimension. The data is split 80-20, meaning 80% of the data is used for training, while the remaining 20% is allocated for validation. The testing was conducted on a few data files previously unseen by the models.

The first dimension (300) in `sensor_data` and `camera_data` refers to the simulation samples through time. Although Isaac Gym operates at 60Hz, our specific task environment is set to sample rate to 6Hz, meaning that each environment yields 6 samples per second. Each episode is run for 50 seconds, resulting in a total of 300 samples.

The second dimension (12) denotes the number of environments per episode. Isaac Gym allows for multiple isolated environments to be run concurrently inside the same episode in order to reduce the rendering overhead, easily noticed in the background of Figure 3.1. We leverage this feature for the collection of our data by running 12 environments per episode.

The remaining dimensions refer to the specific observations and are covered in the subsequent sections.

3.2.1 Sensor Data

As we have seen from the previous section, for each timestep and for each environment we essentially have 35 sensor features, composed of 2 main categories, force sensors and position/orientation.

The Franka Emika Panda robot in our simulation is articulated using a URDF [36] file, which, among other details, specifies the Degrees of Freedom (DOFs) of the robot. Simply put, the DOFs are the joints of the robot, connecting the various rigid bodies together. In our simulation, the robot comprises 9 DOFs, each equipped with a force sensor. These force sensors are the first 9 of the 35 features. According to the Isaac Gym documentation:

> *"Force sensors can be attached to rigid bodies to measure forces and torques experienced at user-specified reference frames. The readings represent the net forces and torques experienced by the parent body, which include external forces, contact forces, and internal forces applied by the solver (e.g., due to joint drives). A body resting on the ground plane will have a net force of zero."*

The remaining 26 features are two buffers of 13 data points for the state of the grip site and the Franka Emika Panda hand. The term "grip site" refers to a specific location or point on the robotic arm (in this case, the Franka robot) where the gripper is intended to interact with objects. The state of each root body is represented using 13 floats: 3 floats for position, 4 floats for quaternion [19], 3 floats for linear velocity, and 3 floats for angular velocity.

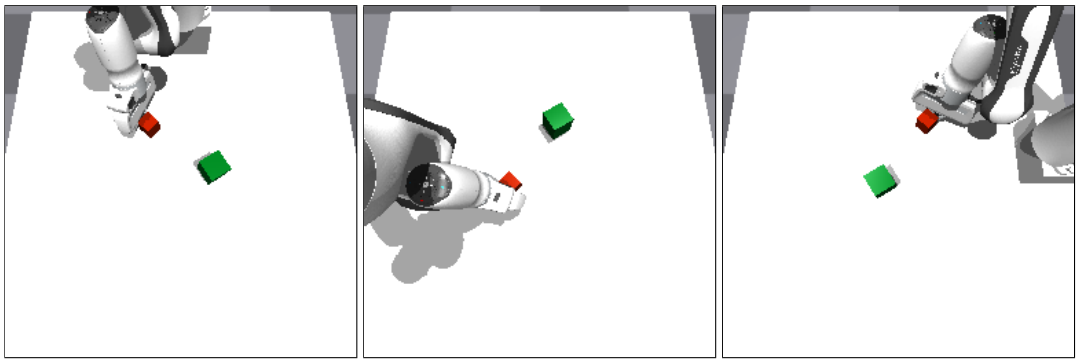
3.2.2 Camera Data

We have placed 3 simulated cameras directly in front, to the right, and to the left of the table, at a slight elevation, pointing at the middle of the table. Figure 3.3 showcases the images taken for a single environment at one timestep. For each timestep, environment, and orientation, we store an RGBA image in tensor format. With 3 orientations, a height and width of 256, and 4 color channels, this results in a shape of `[(3, 256, 256, 4])`.

3.3 Problem Formulation

With a basic understanding of the simulation environment, the data collection task and the dataset, let us now provide the technical details on how we have formulated the problem in order to solve it using the aforementioned model architectures.

An important note for this section, for the sake of simplicity, is that we will assume the camera data can be represented as a one-dimensional tensor with 45 features. The details of how this is done are presented in Section 3.4.



(a) Front (b) Left (c) Right

Figure 3.3: Visualized camera data for one environment in a single timestep

As detailed in Section 3.2, each episode comprises 300 timesteps, incorporating 35 sensor features and an additional 45 image features. When examining a single environment, this results in a time series with 300 samples, each characterized by 80 features.

We aim to use a subset of consecutive timesteps, defined by `window_size`, to predict a subsequent series of timesteps of the same length (`window_size`) that is offset by `prediction_distance` within the total episode. Predicting all 80 features is both unrealistic and unhelpful. Therefore, we design our loss function to compare only a subset of the features, utilizing MSE loss shown in equation 3.1. Figure 3.4 provides a simple illustration on the metric we aim to optimize.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.1}$$

where:

- y_i represents the actual observed value for the i-th data point.
- $\hat{y}_i$ represents the predicted value for the i-th data point.
- n denotes the number of samples the operation is performed on.
- i is the index of the data points, in our case these would be the two sensors.

Figure 3.4 provides an illustration of a toy example with a window size of 4 and a prediction distance of 3. Given the source and target sequences, our objective is to predict features 4 and 5 in the target sequence. We calculate the MSE loss between the predicted and actual values of these features. We refer to the target features as `target_feature_indices`.

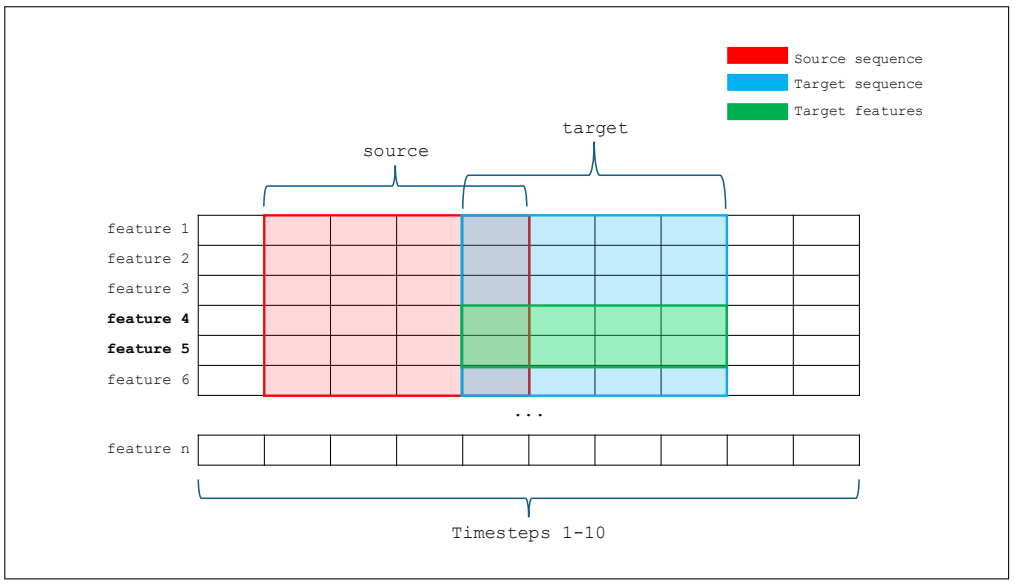


Figure 3.4: Illustration of the problem formulation using a toy example

3.4 Model Implementation

In this section, we will discuss the implementation details of the models used in our study. We will cover the integration of the Transformer, Mamba, and ResNet architectures, explaining how each was adapted and utilized to process our specific dataset.

In terms of technical aspects, we used PyTorch Lightning [10] for the entire development and training process, a framework built on top of the deep learning library PyTorch [33]. The entire codebase is hosted on a GitHub repository [32], and all training was conducted on the Cirrus cluster [9]. Due to the public nature of the Cirrus cluster and its associated limitations on GPU hours, the number of training sessions was restricted, a factor that will be significant in subsequent discussions.

A significant part of what shapes the Transformer into a large language model (LLM) like ChatGPT [31] is its wrappers. These wrappers are essentially the code that surrounds the Transformer, enabling it to become an LLM by handling tasks such as tokenization, preprocessing, and facilitating the training loops. This is why our work in building an efficient wrapper is important.

Given the need to test the two architectures presented in previous sections, we also required a baseline for comparison. For this purpose, we selected the simplest possible model, a basic RNN [28]. Consequently, we performed the same experiments on all three models.

For the implementation of the Transformer and RNN, we built a wrapper around the

official PyTorch implementations, which follow their respective original papers [43, 28]. Our wrapper allows us to train the raw Transformer and RNN architectures using our specific dataset. The same approach was taken for the implementation of the Mamba wrapper; however, the model code was sourced from the official repository provided by the researchers [40].

Naturally, as we have to process camera data, we needed some basic image processing. Unlike the traditional use cases of object classification or detection in image models, our scenario involves integrating sensor and image data. Here, the image data serves as supplementary context rather than the primary focus. Consequently, we determined that training and optimizing a CNN from scratch alongside our other models would be overly complex and would divert focus from the core models we are investigating. Thus, we decided to fine-tune [41] an established image model. We use ResNet-18 [13] by freezing its weights (disabling gradient accumulation) and replacing the last SoftMax layer with a new trainable fully-connected layer. Although ResNet was trained to recognize natural objects, this method allows us to leverage its feature recognition capabilities and only train the last layer to understand the latent representation of the data.

As mentioned in Section 3.2.2, we use 3 images per timestep, per environment. These images are preprocessed and passed through ResNet individually, resulting in a tensor of 15 features per image. The 3 image tensors are then concatenated into a single 45-feature tensor. This final tensor represents the combined latent features of the 3 images, encapsulating the relevant visual information.

Figure 3.5 provides a high level illustration of our architecture with the inputs as discussed in sections 3.2.1 and 3.2.2.

3.5 Training

This section provides an outline of how the model training was conducted and a brief overview of the key parameters.

We treat each episode of a single environment as one training sample. At each training step, we iterate over the episode timesteps, passing the source and target sequences (as shown in Figure 3.4) into the training models. Both windows are then advanced by a number of timesteps equal to the `stride` parameter.

As discussed in Section 3.2.1, the Franka Emika Panda robot in our simulation possesses 9 degrees of freedom, corresponding to its joints. Joints 1 to 7 are depicted in

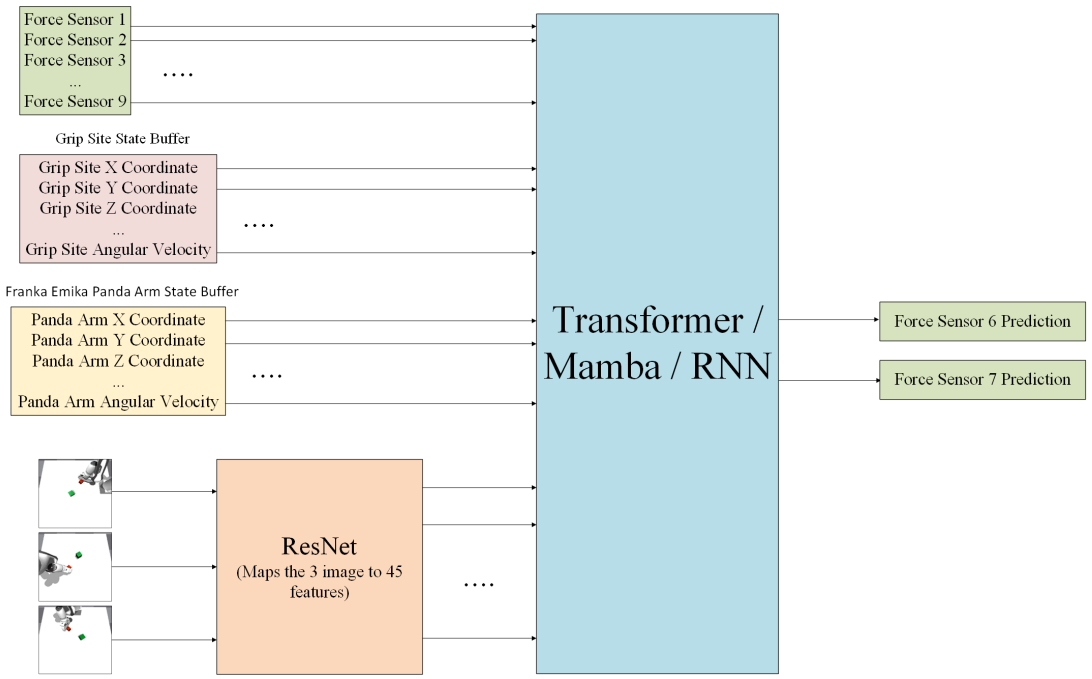


Figure 3.5: Model Architecture

Figure 3.6, while joints 8 and 9 represent the two fingers at the end of the robot's arm.

Our objective is to infer the force sensor output at selected joints. While it is feasible to attempt to infer the force sensor output at all joints, this approach significantly increases the complexity of the problem. Therefore, we have opted to train our models to infer the force sensor output at joints 6 and 7.

These joints are nearest to the end effector and are responsible for fine-tuned control, frequently undergoing changes. The target joints will henceforth be referred to as `target_feature_indices`.

We believe that the sensors closest to the end effector provide the most valuable data and are thus the most worthwhile to model. This is where Equation 3.1 becomes relevant. We will optimize the Mean Squared Error (MSE) between the real and predicted values for joints 6 and 7 throughout an episode.

The Transformer and Mamba models have very similar interfaces, and we control them in an almost identical way. However, the key difference between our two models is that during the Transformer forward pass, we provide both the source and target sequences, whereas Mamba requires only the source sequence. A natural question arises: why do we provide the entire target sequence to the Transformer when we only want to predict a subset of the features? The answer lies in the design of the Transformer architecture. It was originally developed with large language models

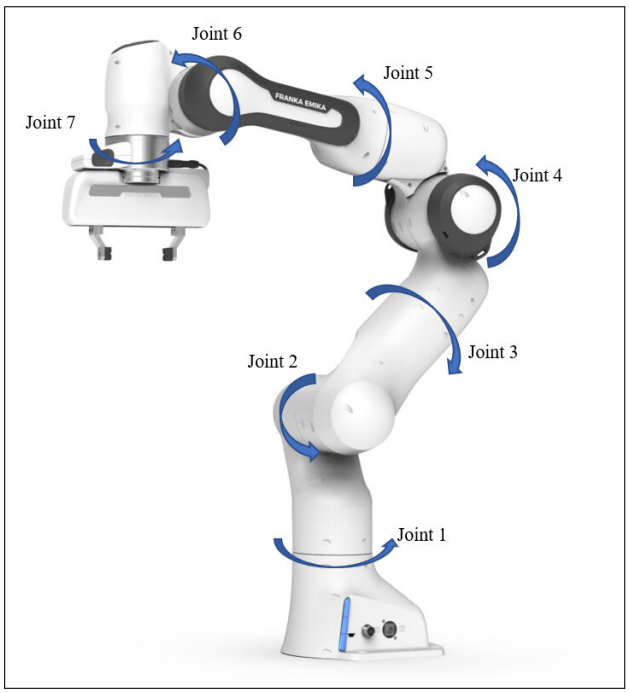


Figure 3.6: Franka Emika Panda arm with labeled joints from [35]

(LLMs) in mind, where in natural language processing (NLP), the source and target sequences generally share the same dimensionality i.e. a sentence in plain English. This is a known limitation of our approach.

We have observed that the robots typically achieve their task (pushing the red cube onto the green) well before the end of the episode, resulting in the robot remaining idle towards the end. Allowing such data could lead our models to frequently predict idleness, yielding poor results. To mitigate this and save computational time, we introduced a parameter, `episode_length`, which defines the number of timesteps to be considered from each episode.

Regarding the data portion, briefly mentioned in 3.2, we set `data_portion` to 0.015, indicating that we utilized 15% of the data, corresponding to 29576 out of 1971744 samples. A sample refers to a batch of inputs that undergoes both a forward and backward pass in the models. In the context of PyTorch Lightning, a forward and backward pass together are referred to as a single step, a term we will use in our experimental discussions.

For the training of our models, we used the well-established Adam optimizer [18] with a linearly decaying learning rate. The Adam optimizer was chosen for its ability to adaptively adjust the learning rates of individual parameters, which has been empirically shown to improve convergence speed and performance in a variety of tasks. The linearly

decaying learning rate helps to prevent the model from overshooting minima during training, facilitating more stable convergence by gradually reducing the learning rate as training progresses. This configuration was selected based on our current experience and best judgment, reflecting empirical evidence and practical considerations in optimizing model performance.

3.6 Experiments

Due to the inherent computational expense of training models for time series problems, combined with the image processing we have integrated and the massive volume of data, our task becomes particularly burdensome. Although we tried various configurations, given our limited computational resources and time constraints, we have chosen to focus on three distinct scenarios, which distill the core of the ideas we want to explore.

- **Experiment 1**: A lightweight scenario utilizing small models with few parameters and minimal context. In this dissertation, the parameter that refers to context is termed `window_size`. The rationale behind this approach is that small models, which encapsulate limited meaning, may perform well under these conditions.

- **Experiment 2**: Conversely, this experiment employs deeper models with a greater number of parameters and a larger `window_size`. The rationale for this approach is that larger models with more parameters and a broader context window can capture more complex patterns and dependencies within the data, potentially leading to improved performance.

- **Experiment 3**: Following the rationale of Experiment 2, we also wanted to try out models that have multiple times the parameter count of the previous experiments but this time we kept the `window_size` the same as in Experiment 2. Given the complexity of the task, larger models are likely necessary to capture it effectively.

Several parameters were maintained consistently across all experiments to ensure a fair comparison. The detailed experiment configurations are listed in Tables A.1, A.2, and A.3. This approach allows for a systematic evaluation of how model complexity and context size affect the performance of the architectures under study.

PyTorch Lightning provides some useful information about models during training, including metrics such as trainable and non-trainable parameters, among others. In the context of model training, ”parameters” refer to the model weights and should

not be confused with configuration parameters. Table A.4 presents an overview of each model's parameters for all three experiments. We aimed to keep the number of parameters relatively similar between models to ensure a fair comparison, targeting approximately 200k parameters in Experiment 1, 600k in Experiment 2, and 10M in Experiment 3. The full parameter breakdown is provided in Table A.4. As evident in Table A.4, each model is comprised of 2 parts, the model we are investigating along with the ResNet block. Notably, the ResNet model has 11.2 million parameters, but as discussed in Section 3.4, most of these are frozen and thus non-trainable.

Model	**Experiment 1**	**Experiment 2**	**Experiment 3**
Transformer	240K	644K	10.2M
Mamba	208K	673K	9.8M
RNN	217K	612K	10.3M

Table 3.1: Number of trainable parameters

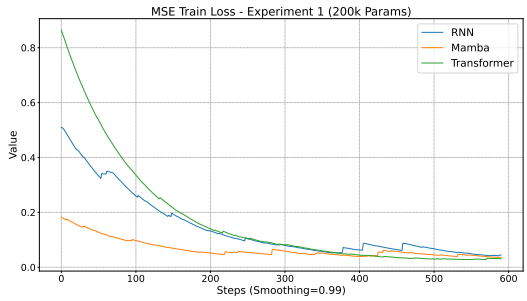
3.7 Results

Let us now present and analyze our experimental results, starting with the training and validation loss curves, followed by an example illustrating the models in action.

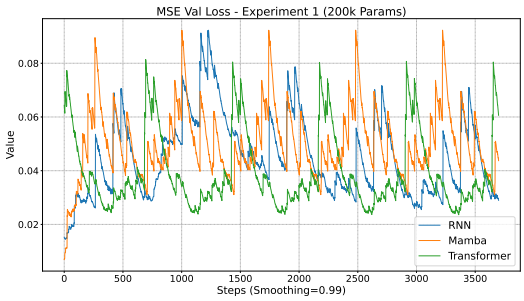
The training and validation MSE curves for Experiments 1 and 2 are presented in Figures 3.7 and 3.8, respectively. In both experiments, all three models (RNN, Mamba, and Transformer) exhibit a significant reduction in training MSE, starting from initial values above 0.5 and gradually decreasing to below 0.1. This indicates effective learning and optimization during the training phase.

However, the validation MSE curves demonstrate considerable volatility. In Experiment 1, the validation MSE fluctuates between 0.01 and 0.08, suggesting potential overfitting or instability in the models' performance on unseen data. In Experiment 2, a notable drop in validation MSE is observed for the RNN and Mamba models, particularly after the initial 500 steps, where the validation loss decreases and stabilizes at lower values. This indicates some improvement in the models' ability to generalize, though fluctuations remain.

When it comes to Experiment 3 (shown in Figure 3.9), we expected the performance to surpass that of the previous two experiments due to the deeper model architecture, which is theoretically better equipped for complex tasks. However, this expectation

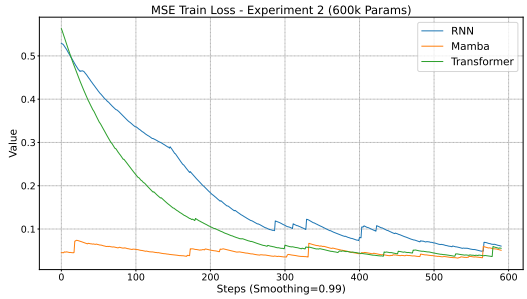


(a) MSE Train Loss

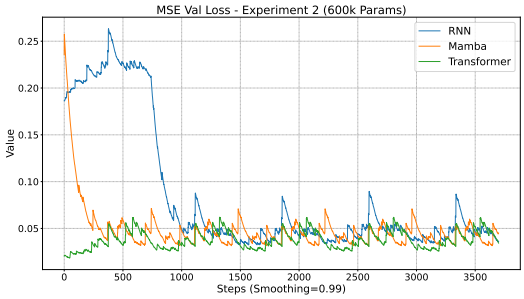


(b) MSE Validation Loss

Figure 3.7: MSE Loss for Experiment 1 (200k Params)



(a) MSE Train Loss



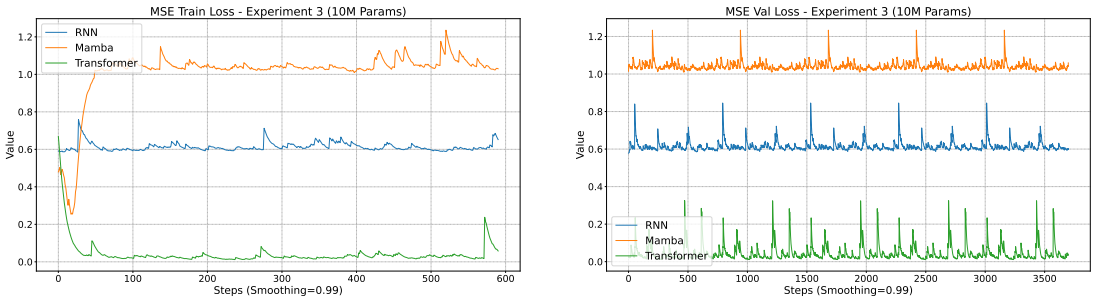
(b) MSE Validation Loss

Figure 3.8: MSE Loss for Experiment 2 (600k Params)

was not met. The performance was significantly worse, with Mamba's training and validation MSE stabilizing around 1.1, RNN's at 0.7, and the Transformer's at 0.1. These results suggest that increasing model complexity did not lead to improved performance.

One possible reason for this outcome could be the increased difficulty in optimizing deeper models. As models grow in complexity, they require more sophisticated training techniques and better initialization strategies to converge effectively. Additionally, deeper models are more prone to issues such as vanishing or exploding gradients, which can impede the learning process and result in suboptimal performance. This indicates that while deeper models have greater potential, they also present more significant training challenges that need to be addressed to realize their full capabilities.

It is important to note that all figures are presented with a smoothing factor of 0.99, utilizing an exponential moving average. Despite the overall downward trend, the unsmoothed raw data curves exhibit significant volatility. For comparison, the raw data curves are also provided (Figures A.1, A.2, A.3). An additional noteworthy point to make is that the number of steps in the graphs varies between training and validation due to technical details such as batch size, which should not be a cause for concern.



(a) MSE Train Loss (b) MSE Validation Loss

Figure 3.9: MSE Loss for Experiment 3 (10M Params)

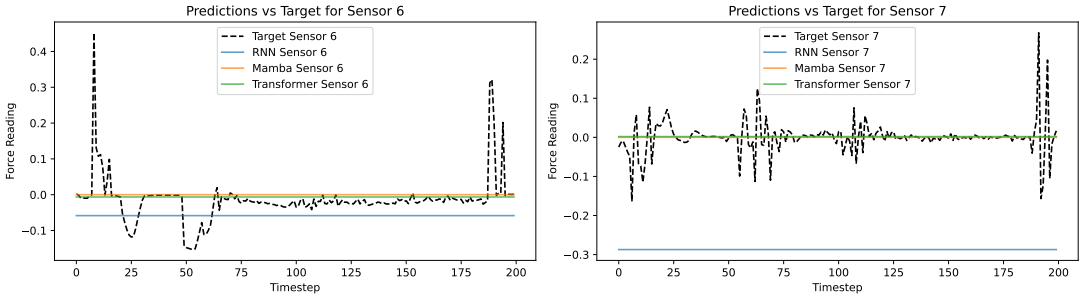


Figure 3.10: Experiment 1 Sensor Inference Illustration

Figures 3.10 and 3.11 illustrate the predictive capabilities of the three models compared to the actual sensory output of sensors 6 and 7 from the same episode. The results show that all three models tend to predict the same value with minimal variation, and this value approximates the average of all sensor readings (approximately 0). In Experiment 1, the Transformer and Mamba models perform similarly, closely predicting the average, while the RNN model is slightly below the average. In Experiment 2, all models closely predict the average.

In Experiment 3 prediction results (Figure 3.12), the Transformer model continues to predict the average accurately. However, the predictions from the RNN and Mamba models deviate significantly from the actual values. We suspect that these discrepancies are due to biases from other episodes or the vanishing/exploding gradient problem [14] present in very deep models. This theory is supported by the fact that the Transformer model, which uses dropout by default, successfully predicts the average while the other models do not. Dropout is a regularization technique used to prevent overfitting by randomly dropping units from the neural network during training [39]. On the contrary, Mamba and RNN that do not incorporate any regularization methods by default are subject to this problem.

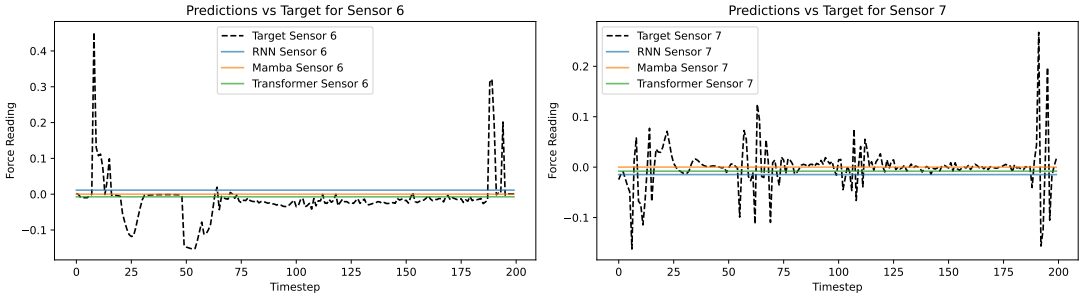


Figure 3.11: Experiment 2 Sensor Inference Illustration

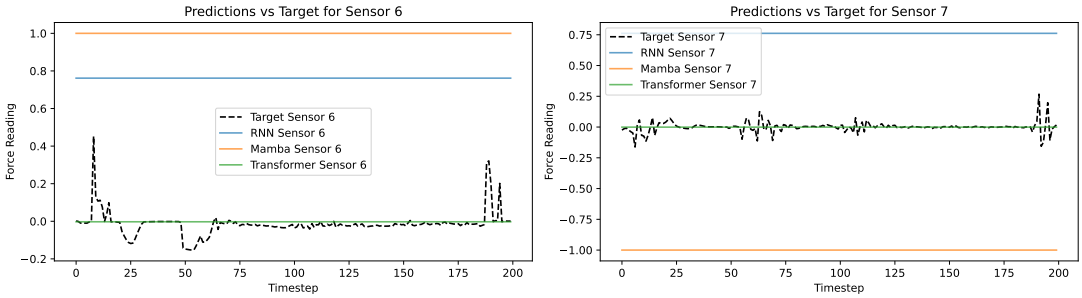


Figure 3.12: Experiment 3 Sensor Inference Illustration

Of course, our goal was not merely to predict the average but to accurately forecast the next sensor output. However, predicting the average is a common issue in regression models and time series problems, often indicating that the models are not complex enough to capture the underlying patterns in the data [27, 7, 20].

Table 3.2 presents several post-training metrics, primarily provided by PyTorch Lightning, including the final training and validation MSE values. The column we would like to highlight, however, is the ”Relative” column, which represents the training time. Due to the nature of their architecture, training these models is extremely time-consuming and poses a significant limitation to our research. A point we will be discussing in the following section.

3.8 Discussion

Following the underwhelming results presented in Section 3.7, we will now discuss their implications concerning the goals we set out to achieve and provide a general outlook on the study as a whole.

The primary objective of this dissertation was to develop a machine learning model capable of inferring sensor outputs from a subset of sensors in a robotic system. This

Run	Value	Step	Relative
Mamba Exp1 200k	0.0041 / 0.002	2,954 / 3,699	13.95 hr / 11.92 hr
Mamba Exp2 600k	0.004 / 0.1065	2,954 / 3,699	13.95 hr / 11.95 hr
Mamba Exp3 10M	1.0149 / 1.2242	2,954 / 3,699	15.8 hr / 13.5 hr
RNN Exp1 200k	0.0683 / 0.011	2,954 / 3,699	16 hr / 13.68 hr
RNN Exp2 600k	0.0081 / 0.0088	2,954 / 3,699	14 hr / 12 hr
RNN Exp3 10M	0.5858 / 0.5459	2,954 / 3,699	13.51 hr / 11.58 hr
Transformer Exp1 200k	0.0022 / 0.001	2,954 / 3,699	14.06 hr / 12.03 hr
Transformer Exp2 600k	0.0149 / 0.0041	2,954 / 3,699	13.15 hr / 11.22 hr
Transformer Exp3 10M	0.0099 / 0.0553	2,954 / 3,699	16.05 hr / 13.7 hr

Table 3.2: Post-training train / validation metrics

goal was pursued with the intention of potentially reducing the number of physical sensors needed, thereby lowering costs and increasing efficiency. However, the results indicate that we have not fully achieved this goal.

The models tested—Transformer, Mamba, and RNN—did not perform as well as expected. While the models were able to predict average sensor values to some extent, they struggled with accurately capturing the nuances of the sensor data. This outcome suggests that the models were not complex enough for the task. Modern large language models (LLMs) ([1, 42, 8]) demonstrate remarkable performance across various domains, typically comprising billions or even trillions of parameters. These models significantly surpass the complexity of those utilized in this study, requiring the computational power of tens or hundreds of GPUs and TPUs for training over several weeks. In layman's terms, our models were not strong enough to solve the problem and instead settled on always guessing the average to avoid the risk of being wrong.

One striking similarity between our problem and a well-known phenomenon in Generative Adversarial Networks (GANs) is mode collapse. Mode collapse occurs when a GAN, instead of generating a diverse set of outputs, repeatedly produces a limited range of results, effectively 'collapsing' into a single mode of output distribution. Similarly, our models seemed to converge on predicting average sensor values, avoiding the complexity of the task at hand. This suggests that, like mode collapse in GANs, our models might be struggling to explore the full distribution of the sensor data, instead settling on a safe but uninformative average prediction [37].

There are several notable drawbacks and challenges observed during this study. One of the major drawbacks was the slow inference speed, which typically lasted between

0.5 and 1 second. Although this might not be significant in some real-time applications, it is problematic for a robotic system that requires readings at rates of 60Hz for example. The need to process large amounts of data, especially images, at each step significantly slows down the inference. This issue is compounded by the complex architectures of the models, making real-time application challenging. Moreover, the task chosen for this study was highly specific. While the models remain conceptually promising, their applicability is limited to very specific robotic tasks. Ideally, these models should be trained on a diverse range of tasks to improve their generalization capabilities.

The problem addressed in this study involved handling data with 80 dimensions, which significantly increased the complexity of the task. Managing such a high-dimensional dataset poses substantial challenges for machine learning models, as each additional dimension can exponentially increase the computational requirements and the risk of overfitting. The high dimensionality makes it difficult for models to capture the intricate relationships between features, leading to a harder and more resource-intensive problem to solve. This is a well known problem in Machine Leanring, often refereed to as *"The curse of dimensionality"*.

The training of these models was extremely time-consuming, with each model requiring extensive computational resources. This highlights the need for significant investment in hardware and time, which may not be feasible for all applications. Additionally, due to time constraints, there was no opportunity for extensive hyperparameter optimization. This limitation likely impacted the models' performance, as fine-tuning could have led to better results.

In hindsight, several aspects could have been approached differently to potentially yield better results. More advanced models with greater complexity might have been employed, and a more comprehensive approach to hyperparameter optimization could have been adopted. Additionally, implementing more robust regularization techniques and addressing the issue of vanishing and exploding gradients from the outset might have improved the models' performance. A comprehensive list of our recommendations is provided in a later section.

Chapter 4

Conclusion

4.1 Summary

This dissertation explored the potential of utilizing modern machine learning techniques, specifically Transformers and Selective State Spaces (Mamba), to replicate and replace physical sensors in robotic systems. The study aimed to determine if virtual sensing could be a viable solution for reducing the costs associated with sensor integration in robots. Despite the growing demand for robotic sensors, the study faced challenges in terms of model complexity, data handling, and computational resources.

The research involved developing a machine learning model capable of inferring sensor outputs from a subset of sensors in a robotic system. Data was collected through a simulated environment where a Franka Emika Panda robot performed a specific task. The collected data included various measurements, such as positions, orientations, and images from cameras, which were then used to train and evaluate the models.

Three experiments were conducted to test the efficacy of the models under different configurations. These experiments varied in terms of model complexity and context size, with the goal of understanding the impact of these factors on model performance. The results showed that while the models could predict average sensor values, they struggled to capture the nuances of the sensor data. The Transformer model demonstrated some potential, but overall, the models did not perform as well as expected.

The study highlighted several challenges, including the slow inference speed, the high dimensionality of the data, and the time-consuming nature of training these models. These challenges underscored the need for more advanced models, better training techniques, and optimized data handling strategies.

In conclusion, while the research did not fully achieve its objectives, it provided

valuable insights into the complexities of virtual sensing in robotics. Future work should focus on employing more complex models, reducing data dimensionality, and exploring techniques to speed up inference and improve generalization capabilities. The knowledge gained from this study lays the foundation for further exploration and development in the field of virtual sensing.

4.2 Future Work

During this study, we identified several pitfalls and challenges. Here, we present a series of recommendations for future research aimed at improving model performance, efficiency, and applicability in the field of virtual sensing for robotics.

- Employing more complex models with higher parameter counts, similar to modern LLMs, could improve performance.
- Implementing techniques such as Principal Component Analysis (PCA) [16] to reduce the dimensionality of the data could help mitigate the curse of dimensionality and improve the efficiency of the models.
- Training models on a broader range of tasks would likely improve their generalization capabilities, making them more applicable to a variety of robotic applications.
- Techniques to speed up inference, such as optimized image processing and more efficient model architectures, should be explored.
- Further research into regularization techniques and strategies to mitigate the vanishing and exploding gradient problem is necessary.
- A thorough hyperparameter optimization process should be undertaken to fine-tune the models for better performance.

Although these recommendations address the problems we encountered, they are likely insufficient, and further research and experimentation are needed.

4.3 Closing Remarks

Overall, while this study did not fully achieve its primary objectives, it provides valuable insights into the challenges and potential solutions in the field of virtual sensing for

robotics. By addressing the identified limitations and exploring the suggested future work, significant advancements can be made towards developing more efficient and effective machine learning models for robotic sensor inference.

Although we may not have fully achieved our initial goal, the invaluable experience and knowledge we gained along the journey are undoubtedly more significant.

"Maybe the real treasure was the friends we made along the way."

— Unknown

Bibliography

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Roberto Calandra, Andrew Owens, Dinesh Jayaraman, Justin Lin, Wenzhen Yuan, Jitendra Malik, Edward H Adelson, and Sergey Levine. More than a feeling: Learning to grasp and regrasp using vision and touch. *IEEE Robotics and Automation Letters*, 3(4):3300–3307, 2018.

[4] Roberto Calandra, Andrew Owens, Manu Upadhyaya, Wenzhen Yuan, Justin Lin, Edward H Adelson, and Sergey Levine. The feeling of success: Does touch sensing help predict grasp outcomes? *arXiv preprint arXiv:1710.05512*, 2017.

[5] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[6] Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality. *arXiv preprint arXiv:2405.21060*, 2024.

[7] Derek J De Solla Price et al. Predicting the average number of citations per paper of a research group. *Nature*, 205:415–416, 2006.

[8] Google DeepMind. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[9] Edinburgh Parallel Computing Centre. Epcc facilities: Cirrus. https://www.epcc.ed.ac.uk/facilities/cirrus, 2024. Accessed: 2024-07-28.

[10] William Falcon and et al. Pytorch lightning. https://github.com/PyTorchLightning/pytorch-lightning, 2019.

[11] Franka Emika. Franka emika: Robot system. https://franka.de/, 2024.

[12] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[14] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[15] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[16] Ian T. Jolliffe. *Principal Component Analysis*. Springer, 2nd edition, 2002.

[17] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. In *International Conference on Machine Learning*, pages 2410–2419. PMLR, 2018.

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[19] Jack B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princeton University Press, Princeton, NJ, 1999.

[20] Vinay Kumar, UK Paul, and Vikrant Yadav. Predictive modeling of rainfall using data mining and artificial intelligence techniques. *International Journal of Advance Research, Ideas and Innovations in Technology*, 4(2):405–410, 2018.

[21] Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165, 2017.

[22] Michelle A Lee, Matthew Tan, Yuke Zhu, and Jeannette Bohg. Detect, reject, correct: Crossmodal compensation of corrupted sensors. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 909–916. IEEE, 2021.

[23] Michelle A Lee, Yuke Zhu, Krishnan Srinivasan, Parth Shah, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8943–8950. IEEE, 2019.

[24] Michelle A Lee, Yuke Zhu, Peter Zachares, Matthew Tan, Krishnan Srinivasan, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Jeannette Bohg. Making sense of vision and touch: Learning multimodal representations for contact-rich tasks. *IEEE Transactions on Robotics*, 36(3):582–596, 2020.

[25] Li Liu, Fu Guo, Zishuai Zou, and Vincent G Duffy. Application, development and future opportunities of collaborative robots (cobots) in manufacturing: A literature review. *International Journal of Human–Computer Interaction*, 40(4):915–932, 2024.

[26] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[27] Spyros Makridakis and Michele Hibon. The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of Forecasting*, 1(2):111–153, 1982.

[28] Larry R Medsker, Lakhmi Jain, et al. Recurrent neural networks. *Design and Applications*, 5(64-67):2, 2001.

[29] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to Linear Regression Analysis*. John Wiley & Sons, 2012.

[30] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[31] OpenAI. Chatgpt: A large language model trained by openai. `https://www.openai.com/chatgpt`, 2023.

[32] Loucas Papalazarou. Virtual sensing. `https://github.com/loucaspapalazarou/virtual-sensing`, 2024.

[33] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[34] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019.

[35] Amit Rogel, Richard Savery, Ning Yang, and Gil Weinberg. Robogroove: Creating fluid motion for dancing robotic arms. In *Proceedings of the 2022 ACM/IEEE International Conference on Human-Robot Interaction*, pages 1–9, 06 2022.

[36] Radu Bogdan Rusu and Sachin Chitta. Urdf: Unified robot description format. `http://wiki.ros.org/urdf`, 2019. Accessed: 2024-07-19.

[37] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[38] Mohsen Soori, Behrooz Arezoo, and Roza Dastres. Artificial intelligence, machine learning and deep learning in advanced robotics, a review. *Cognitive Robotics*, 2023.

[39] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[40] State Spaces Team. Mamba: State spaces. `https://github.com/state-spaces/mamba`, 2024.

[41] PyTorch Team. Torchvision object detection finetuning tutorial. `https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html`, 2024. Accessed: 2024-07-28.

[42] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Appendix A

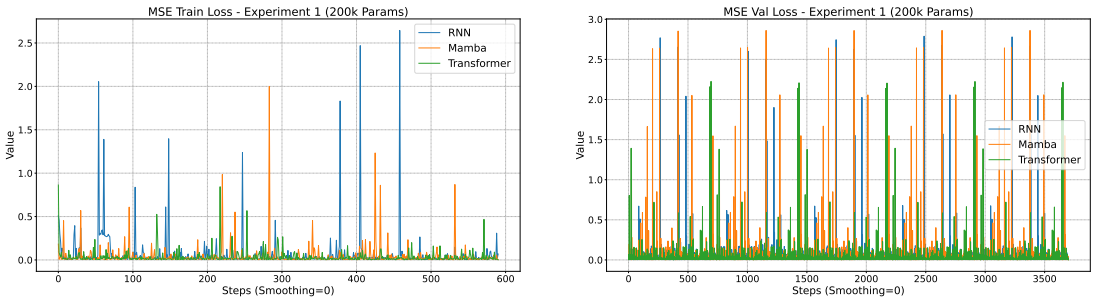
Supplementary Material

Parameter	Value
data_dir	data-w-camera
resnet_checkpoint	resnet18-f37072fd.pth
data_portion	0.015
episode_length	200
batch_size	8
num_workers	0
max_epochs	1
prediction_distance	1
window_size	10
start_lr	0.005
end_lr	0.001
activation	tanh
stride	1
log_every_n_steps	1
target_feature_indices	6,7
resnet_features	15
transformer_num_encoder_layers	2
transformer_num_decoder_layers	2
transformer_nhead	4
transformer_dim_feedforward	128
mamba_d_state	128
mamba_d_conv	16
mamba_expand	4
rnn_rnn_hidden_size	200
rnn_num_layers	3

Table A.1: Experiment 1 Configuration

Parameter	Value
data_dir	data-w-camera
resnet_checkpoint	resnet18-f37072fd.pth
data_portion	0.015
episode_length	200
batch_size	8
num_workers	0
max_epochs	1
prediction_distance	1
window_size	30
start_lr	0.005
end_lr	0.001
activation	tanh
stride	1
log_every_n_steps	1
target_feature_indices	6,7
resnet_features	15
transformer_num_encoder_layers	4
transformer_num_decoder_layers	4
transformer_nhead	8
transformer_dim_feedforward	256
mamba_d_state	256
mamba_d_conv	32
mamba_expand	8
rnn_rnn_hidden_size	256
rnn_num_layers	5

Table A.2: Experiment 2 Configuration



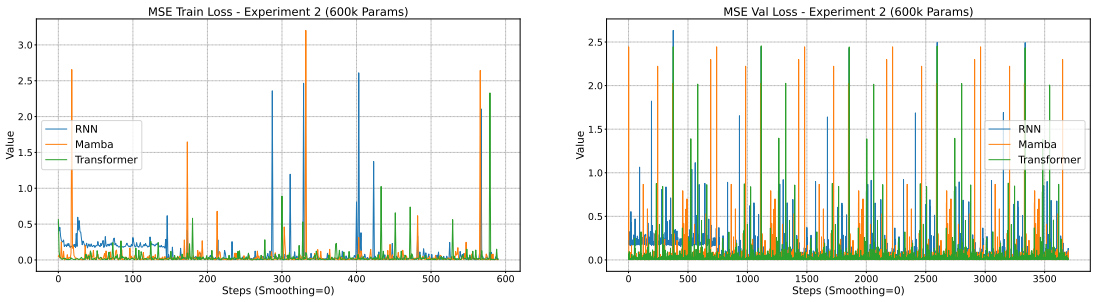
(a) MSE Train Loss

(b) MSE Validation Loss

Figure A.1: MSE Loss for Experiment 1 (200k Params)

Parameter	Value
data_dir	data-w-camera
resnet_checkpoint	resnet18-f37072fd.pth
data_portion	0.015
episode_length	200
batch_size	8
num_workers	0
max_epochs	1
prediction_distance	1
window_size	10
start_lr	0.005
end_lr	0.001
activation	tanh
stride	1
log_every_n_steps	5
target_feature_indices	6,7
resnet_features	15
transformer_num_encoder_layers	25
transformer_num_decoder_layers	25
transformer_nhead	20
transformer_dim_feedforward	1024
mamba_d_state	256
mamba_d_conv	512
mamba_expand	80
rnn_rnn_hidden_size	512
rnn_num_layers	20

Table A.3: Experiment 3 Configuration



(a) MSE Train Loss

(b) MSE Validation Loss

Figure A.2: MSE Loss for Experiment 2 (600k Params)

Name	**Type**	**# of Params**
Transformer		
resnet	ResNetBlock	11.2 M
model	Transformer	**240 K**
247 K Trainable params		
11.4 M Total params		
Mamba		
resnet	ResNetBlock	11.2 M
model	Mamba	**208 K**
216 K Trainable params		
11.4 M Total params		
RNN		
resnet	ResNetBlock	11.2 M
model	RNN	**217 K**
224 K Trainable params		
11.4 M Total params		

Experiment 1

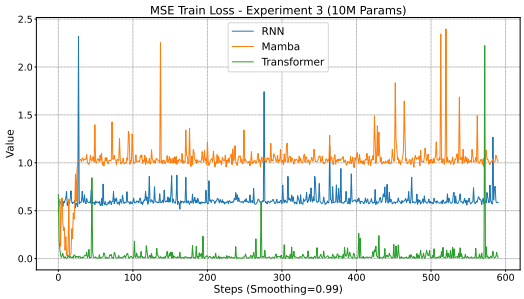
Name	**Type**	**# of Params**
Transformer		
resnet	ResNetBlock	11.2 M
model	Transformer	**644 K**
652 K Trainable params		
11.8 M Total params		
Mamba		
resnet	ResNetBlock	11.2 M
model	Mamba	**673 K**
681 K Trainable params		
11.9 M Total params		
RNN		
resnet	ResNetBlock	11.2 M
model	RNN	**612 K**
620 K Trainable params		
11.8 M Total params		

Experiment 2

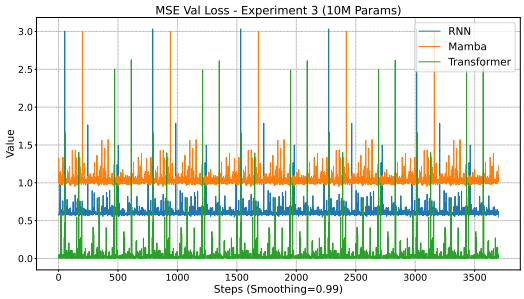
Name	**Type**	**# of Params**
Transformer		
resnet	ResNetBlock	11.2 M
model	Transformer	**10.2 M**
10.2 M Trainable params		
11.8 M Total params		
Mamba		
resnet	ResNetBlock	11.2 M
model	Mamba	**673 K**
681 K Trainable params		
11.9 M Total params		
RNN		
resnet	ResNetBlock	11.2 M
model	RNN	**612 K**
620 K Trainable params		
11.8 M Total params		

Experiment 3

Table A.4: Model Parameters



(a) MSE Train Loss



(b) MSE Validation Loss

Figure A.3: MSE Loss for Experiment 3 (10M Params)