LanSAR – Language-commanded Scene-aware Action Response

by

Adam Nelson Hardy

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2024 by the
Graduate Supervisory Committee:

Heni Ben Amor, Chair
Siddharth Srivastava
Theodore Pavlic

ARIZONA STATE UNIVERSITY

May 2024

ABSTRACT

Robot motion and control remains a complex problem both in general and in the field of machine learning (ML). Without ML approaches, robot controllers are typically designed manually, which can take considerable time, generally requiring accounting for a range of edge cases and often producing models highly constrained to specific tasks. ML can decrease the time it takes to create a model while simultaneously allowing it to operate on a broader range of tasks. The utilization of neural networks to learn from demonstration is, in particular, an approach with growing popularity due to its potential to quickly fit the parameters of a model to mimic training data.

Many such neural networks, especially in the realm of transformer-based architectures, act more as planners, taking in an initial context and then generating a sequence from that context one step at a time. Others hybridize the approach, predicting a latent plan and conditioning immediate actions on that plan. Such approaches may limit a model's ability to interact with a dynamic environment, needing to replan to fully update its understanding of the environmental context. In this thesis, Language-commanded Scene-aware Action Response (LanSAR) is proposed as a reactive transformer-based neural network that makes immediate decisions based on previous actions and environmental changes. Its actions are further conditioned on a language command, serving as a control mechanism while also narrowing the distribution of possible actions around this command. It is shown that LanSAR successfully learns a strong representation of multimodal visual and spatial input, and learns reasonable motions in relation to most language commands. It is also shown that LanSAR can struggle with both the accuracy of motions and understanding the specific semantics of language commands.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Learning from demonstration (LfD) has become a pivotal methodology in the field of robotics, allowing a model to learn the motions and actions required to accomplish tasks via observation of real or simulated tasks. In the context of robot arm control, a model would learn the intricate motions required to execute tasks and manipulate objects in a range of scenarios, as opposed to needing to be directly programmed with such actions, which often results in a model constrained to highly specific tasks. As such, with sufficient data and sufficient architecture, a more flexible model can be created in substantially less time.

In more recent years, the integration of transformer architectures into the domain of LfD has shown great promise in the field of robotics. Originally developed for natural language processing (NLP) by Vaswani *et al.* (2017), transformers are deep learning models that are capable of learning long-term dependencies of sequential data in parallel by leveraging self-attention mechanisms, making them ideal candidates for interpreting temporal sequences.

NLP, in particular, has been an area of interest in the field of robotics, with the goal of utilizing machine learning to train robots to perform tasks conditioned on natural language commands. Not only can language commands provide a simple way of directing a robot to perform a task, but they can also assist in the learning routine by constraining a model's otherwise large action space to a more limited distribution of actions corresponding to the specified command. This, in turn, can aid in allowing a model to learn a wide range of possible tasks rather than focusing on a specific task at a time.

Learning to map a complex array of multimodal input data, such as visual and spatial information, to actions is not a simple task, hence LfD being an ongoing area of research in robotics. This thesis will discuss an approach for attempting to solve this problem via the utilization of LanSAR, a multimodal transformer architecture that is capable of generating motion based on a language command such as, "open the drawer," a history of previous actions, and changes in the environment over time. The output action space of LanSAR consists of 3D position and orientation deltas, as well as a binary gripper action. Chapter 2 will provide further details on the specifics of general transformer networks. Chapter 3 will discuss the architecture and implementation of LanSAR. Chapter 4 will detail all materials and methodologies used to train and evaluate LanSAR. Chapter 5 will detail the results of the experiments described in chapter 4. Chapter 6 will analyze and discuss the results of these experiments, detailing the reasons LanSAR performs at the level it does, and then discuss the future potential of this model and its architecture.

1.1 Related Works

There have been numerous approaches to modeling motion in the field of machine learning, especially in relation to NLP. Lynch and Sermanet (2021) propose Multi-context imitation learning (MCIL), which predicts robot motion based on perceptual data and a latent plan formed from the same perceptual information as well as a language goal. Mees *et al.* (2022a) extend MCIL with Hierarchical Universal Language Conditioned Policies (HULC), which incorporates gripper image data and attempts to address the issue of similar language encodings for similar commands via a contrastive learning approach. Both architectures act as hybridized planners, in which real-time decisions are made based on their latent plans and perceptual information. This lessens the potential for error that pure planner-based models face but, if the latent

plan is suboptimal, then immediate decisions conditioned on this plan may also be suboptimal until the model replans.

Moving away from the MCIL line of robot motion models, Wu *et al.* (2023) propose GR-1, a transformer architecture that is pre-trained with a large motion dataset before being fine-tuned on a task-specific dataset, and is capable of jointly generating both visual data and robot motion. GR-1's performance is impressive but hinges on a two-phase training routine, each with its own set of hyperparameters, with additional complexity due to its image-generative auxiliary task.

Ke *et al.* (2024) leverage a diffusion approach with 3D Diffuser Actor, which jointly learns a 3D representation of a scene and learns to diffuse noisy predictions of robot motion into a smooth trajectory. Zhang *et al.* (2024) propose Language Control Diffusion (LCD), a model that acts similarly to MCIL, generating a latent plan, but then also employs diffusion techniques to denoise the plan before making a final motion prediction. While performant, diffusion models can introduce additional complexity to the problem in the form of determining optimal noise sampling techniques and denoising iterations, as opposed to directly modeling trajectory dependencies.

Other approaches such as Zhou *et al.* (2024) and Driess *et al.* (2023) try to separate robot control into different levels of policies, with the former leveraging large language models (LLMs) to generate high-level plans which are then used to condition a low-level motion policy. The latter jointly learns an intermediate policy for specific skills such as "rotation," "transportation," and, "grasping," and executes a low-level policy based on the selected skill. Mees *et al.* (2023) extend MCIL and HULC yet further with HULC++, which learns a high-level affordance policy that estimates a goal location and moves the robot arm to the general desired location before letting a learned low-level policy perform the more precise interactions needed. This hierarchical separation of policies can potentially yield highly performant results

but suffers from a similar issue as hybridized planner models, in which suboptimal high-level policies can negatively impact low-level policies.

While not intended to solve robot motion, Seff *et al.* (2023) propose MotionLM, which closely aligns with LanSAR's architecture via its utilization of transformer architectures to encode spatial and temporal data. MotionLM encodes spatial and environment information using a permutation invariant set transformer (Lee *et al.*, 2019), which can learn a holistic encoding representing an unordered sequence of data and uses this encoding as the memory sequence to the transformer's decoder to predict a horizon of future states for agents in a scene by outputting a discretized set of motion deltas. While MotionLM demonstrates strong predictive capabilities, discretizing the action space inherently decreases the granularity of information.

Also unintended for robot motion, Zhou *et al.* (2019) propose Swarmnet, which attempts to solve a similar problem of predicting motion based on spatiotemporal information, in this case for a multiagent scenario. Swarmnet first encodes a window of temporal information using 1D convolutions. It then leverages a graph neural network (GNN) to encode unordered spatial information by modeling the temporal encodings of agents as nodes in a graph and their interactions as edges. Swarmnet demonstrates a strong capability in understanding spatial relationships but struggled to learn long-term motion dynamics, requiring a curriculum learning routine that substantially increased training time.

LanSAR aims to act entirely as a low-level reactive policy, with the goal of making real-time decisions in continuous space based on its current state and previous states, immediate and previous perceptual information, and a language command, potentially avoiding the issues of plan-based and hierarchical models. LanSAR can also be easily trained with a single-phase routine and a fairly small amount of data, simply minimizing motion and gripper action losses for each step of training.

Chapter 2

TRANSFORMER ARCHITECTURE

This chapter serves to provide an in-depth overview of transformer models, closely following the implementation and theoretical concepts introduced by Vaswani *et al.* (2017). LanSAR is fundamentally a transformer architecture, thus understanding transformers is essential to understanding LanSAR's implementation and learning dynamics. Readers who are familiar with encoders, decoders, and attention mechanisms may move on to chapter 3, where the implementation of LanSAR's architecture is defined.

2.1 Transformer Overview

Transformers, at their most basic level, typically consist of an encoder and decoder network, the former of which takes in an input sequence and the latter of which takes in a target sequence, both of which are usually sequences of high-dimensional, latent features projected from lower dimensionality via either embedding or linear layers. Positional encodings are added to preserve a sense of order in the sequence, which the transformer does not otherwise account for. A general example of a transformer architecture can be seen in figure 2.1.

2.2 Encoder

Each encoder layer consists of a self-attention mechanism and a feed-forward network. Given an input sequence X consisting of $[x_1, x_2, \ldots, x_n]$ where each x_i is a continuous value or discrete token, these values are first projected to a higher dimensional space via either linear layer(s) in the case of continuous values or an embedding

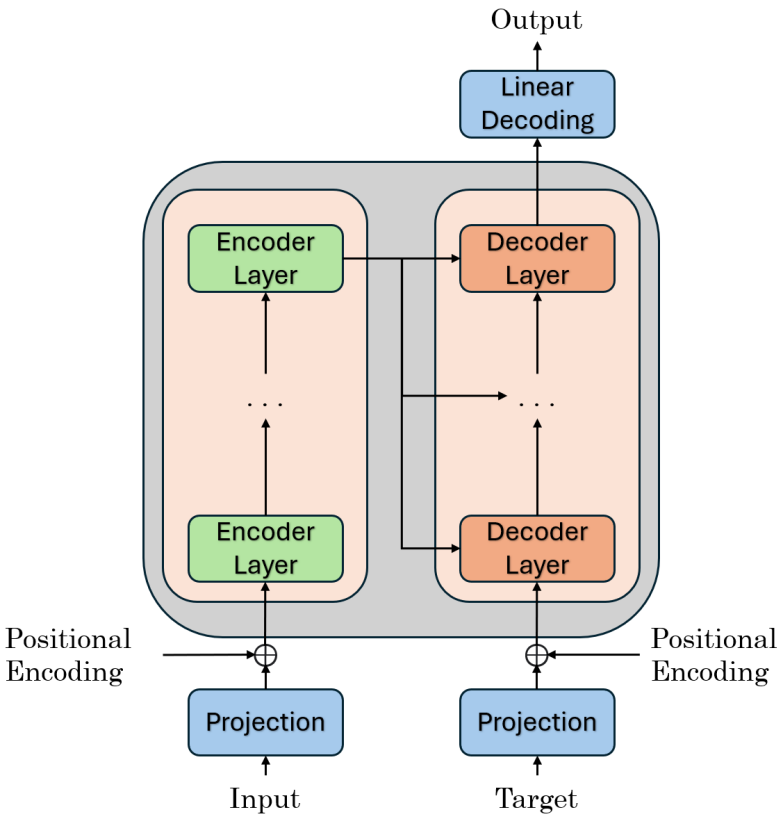


Figure 2.1: Example of General Transformer Architecture with a Stack of N Encoder Layers and N Decoder Layers.

layer in the case of discrete values.

$$X^{\text{proj}} = E_{\text{input}}(X) \tag{2.1}$$

A positional encoding p_i is computed via an embedding layer for each index in the sequence of length N and then added to the projected values in the sequence.

$$P^{\text{input}} = [P^{\text{input}}_{\text{emb}}(i),\ i \in \{0, 1, ..., n-1\}] \tag{2.2}$$

$$X^{\text{enc}} = [x_i^{\text{proj}} + p_i^{\text{input}},\ i \in \{0, 1, ..., n-1\}] \tag{2.3}$$

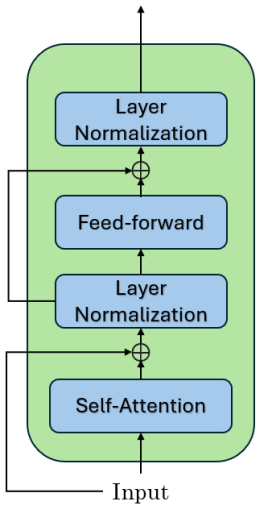


Figure 2.2: Transformer Encoder Layer Visualization

Self-attention is utilized on the full sequence X^{enc} by projecting it to a query Q, key K, and value V via learnable linear transformations.

$$Q = Q_L^{\text{input}}(X^{\text{enc}}) \tag{2.4}$$

$$K = K_L^{\text{input}}(X^{\text{enc}}) \tag{2.5}$$

$$V = V_L^{\text{input}}(X^{\text{enc}}) \tag{2.6}$$

The attention for attention head h_i is calculated as,

$$h_i = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.7}$$

where $\sqrt{d_k}$ is the dimensionality of the key vectors. $\frac{QK^T}{\sqrt{d_k}}$, after normalization, produces a matrix of attention weights, where each row corresponds to a query and each column corresponds to a key. The value at each position in the matrix represents the importance that the query (row) places on the corresponding key (column). These values serve as weights representing the extent to which the keys in the sequence will contribute to the output at the respective query's position, providing a contextual

relationship across the entire sequence. This matrix is multiplied by the value matrix V, which produces the weighted output for a given attention head. Multiple attention heads are used to allow the attention mechanism to focus on different relationships in the sequence and the outputs of each attention head are concatenated prior to using another linear function to produce the final attention output.

$$A^{\text{input}} = A_L^{\text{input}}(\text{concat}(h_0, h_1, \ldots, h_{h-1})) \tag{2.8}$$

This output is added to the original input as a residual connection to help gradients better flow during training.

$$X^{\text{enc}} = \text{layernorm}(X^{\text{enc}} + A^{\text{input}}) \tag{2.9}$$

This is then sent through a feed-forward neural network, typically set of two linear transformations with a nonlinear activation, and added as yet another residual connection before being normalized again.

$$X^{\text{enc}} = \text{layernorm}(X^{\text{enc}} + F_{\text{input}}(X^{\text{enc}})) \tag{2.10}$$

An example of an encoder layer can be seen in figure 2.2.

2.3 Decoder

As with the encoder, the positional encodings are added to the input to the decoder, denoted as the target sequence $T = [t_0, t_1, \ldots, t_{n-1}]$. Typically, separate linear functions or embedding layers are used for the target sequence from the input sequence.

$$T^{\text{proj}} = E_{\text{target}}(T) \tag{2.11}$$

$$P^{\text{target}} = [P_{\text{emb}}^{\text{target}}(i),\ i \in \{0, 1, ..., n-1\}] \tag{2.12}$$

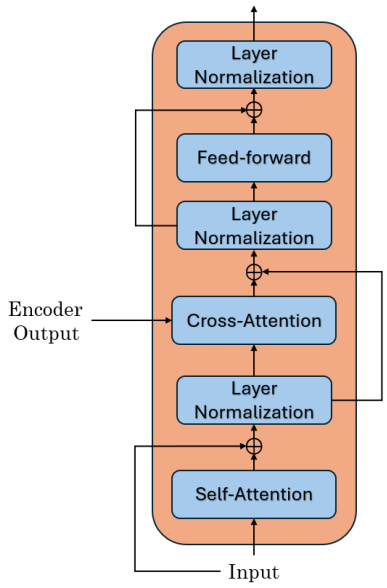


Figure 2.3: Transformer Decoder Layer Visualization

$$T^{\text{enc}} = [t_i^{\text{proj}} + p_i^{\text{target}},\ i \in \{0, 1, ..., n-1\}] \tag{2.13}$$

The decoder, visualized in figure 2.3, primarily differs from the encoder in that it introduces a cross-attention mechanism that utilizes the output from the encoder as a "memory" sequence, allowing the decoder to attend to both the target and encoded input sequence simultaneously. The process begins similarly with a self-attention layer using the target sequence as its input.

$$Q = Q_{L_1}^{\text{target}}(T^{\text{enc}}) \tag{2.14}$$

$$K = K_L^{\text{target}}(T^{\text{enc}}) \tag{2.15}$$

$$V = V_L^{\text{target}}(T^{\text{enc}}) \tag{2.16}$$

$$h_i = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.17}$$

$$A^{\text{target}_1} = A_{L_1}^{\text{target}}(\text{concat}(h_0 h_1, \ldots, h_{h-1})) \tag{2.18}$$

$$T^{\text{enc}} = \text{layernorm}(T^{\text{enc}} + A^{\text{target}_1}) \tag{2.19}$$

From here, cross-attention is utilized between the memory sequence from the encoder and the output from the decoder's self-attention layer. In this case, the target is projected to serve as the queries while the memory is projected to serve as the keys and values.

$$Q = Q_{L_2}^{\text{target}}(T^{\text{enc}}) \tag{2.20}$$

$$K = K_{L}^{\text{memory}}(X^{\text{enc}}) \tag{2.21}$$

$$V = V_{L}^{\text{memory}}(X^{\text{enc}}) \tag{2.22}$$

$$h_i = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.23}$$

The remainder of the decoder effectively functions identically to the encoder.

$$A^{\text{target}_2} = A_{L_2}^{\text{target}}(\text{concat}(h_1, h_2, \ldots, h_n)) \tag{2.24}$$

$$T^{\text{enc}} = \text{layernorm}(T^{\text{enc}} + A^{\text{target}_2}) \tag{2.25}$$

$$T^{\text{enc}} = \text{layernorm}(T^{\text{enc}} + F_{\text{target}}(T)^{\text{enc}}) \tag{2.26}$$

In this thesis, PyTorch's (Paszke *et al.*, 2019) encoder and decoder implementations are used, which closely follow the architecture of Vaswani *et al.* (2017) while allowing the use of the library's performance optimizations and flexibility.

Chapter 3

LANGUAGE-COMMANDED SCENE-AWARE ACTION RESPONSE

LanSAR specifically aims to operate on simulated data of a robot actuator performing a variety of tasks on a desk, such as moving blocks in a number of different ways, opening drawers, and pressing buttons, all aligned with a given natural language command. These otherwise simple tasks require accurate and intricate motions, which means the problem of mapping the language command and other input data to an action is nontrivial.

The overall problem can be summarized as a function approximating a mapping from a command C, sequence of robot states X, block states B, miscellaneous scene information S, static image data I, and gripper image data G, to an end effector delta position Δ_p, delta orientation Δ_o, and gripper action A. The miscellaneous scene information consists of drawer, slider, and button hinge states, as well as binary light states.

$$\Delta_p, \Delta_o, A = F(C, X, B, S, I, G) \tag{3.1}$$

3.1 Model

In this section, a high-level overview of the input data and the approach for modeling the approximation of the aforementioned function is described.

3.1.1 Inputs

At the highest level, LanSAR takes in a sequence of multimodal perceptual information and robot states, and a corresponding language command. This multimodal perceptual information includes the images, block states, miscellaneous scene informa-

tion, and robot states. Images consist of RGB images from both a global perspective and the perspective of a robot arm's end effector. The block states consist of the 3D position and orientation of a red, blue, and pink block. The miscellaneous scene information consists of the joint states of a drawer, sliding door, button, and lever, as well as the binary states of two lights. The robot positions consist of the 3D position and orientation of the end effector, the joint states of the arm, and the joint state of the gripper.

3.1.2 Architecture Overview

LanSAR, seen in figure 3.1, utilizes a pre-trained CLIP module, a multimodal transformer-based model introduced by Radford *et al.* (2021) jointly trained to understand both images and text. It also consists of a series of multilayer perceptrons (MLPs) for projecting all input modalities to the same dimensionality, a context encoder, a motion decoder, and two output MLPs. The context encoder generates a holistic representation of multimodal contextual information, the motion decoder utilizes this context in tandem with previous robot states and conditions on a language command to predict a latent action, which is decoded into an executable motion and gripper action via the two output heads.

3.1.3 Input Layers

A temporal sequence of images and a corresponding language command are first pre-encoded utilizing CLIP. These precomputed encodings, agent states (3D position, orientation, joint states, binary gripper action), block states (3D position, orientation), and miscellaneous scene information are all projected to the transformer's expected dimensionality via MLPs.

$$C^{\text{clip}} = \text{CLIP}(C) \tag{3.2}$$

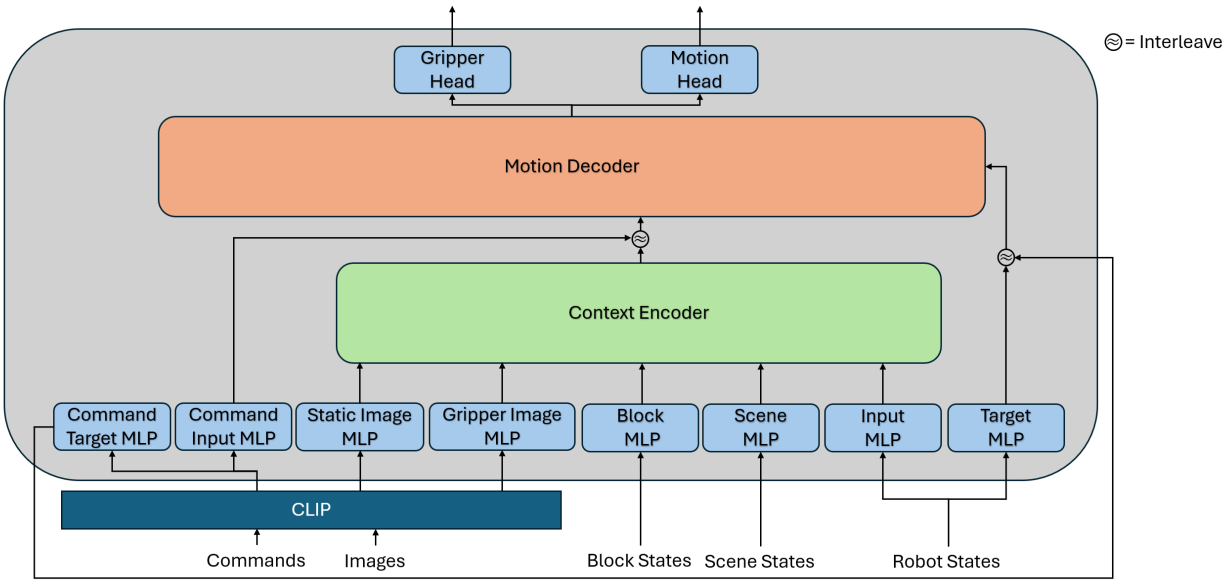


Figure 3.1: LanSAR Architecture Overview

$$I^{\text{clip}} = \text{CLIP}(I) \tag{3.3}$$

$$G^{\text{clip}} = \text{CLIP}(G) \tag{3.4}$$

$$C^{\text{input}} = M_c^{\text{input}}(C^{\text{clip}}) \tag{3.5}$$

$$C^{\text{target}} = M_c^{\text{target}}(C^{\text{clip}}) \tag{3.6}$$

$$I^{\text{input}} = M_{\text{I}}(I^{\text{clip}}) \tag{3.7}$$

$$G^{\text{input}} = M_g(G^{\text{clip}}) \tag{3.8}$$

$$B^{\text{input}} = M_b(B) \tag{3.9}$$

$$S^{\text{input}} = M_h(S) \tag{3.10}$$

$$X^{\text{input}} = M_x^{\text{input}}(X) \tag{3.11}$$

$$X^{\text{target}} = M_x^{\text{target}}(X) \tag{3.12}$$

3.1.4 Context Encoder

The context encoder is intended to learn a holistic encoding of a multimodal sequence, providing a condensed but rich representation of the environment. For a given timestep i, a sequence R is formed from the projected context information $[X^{\text{input}}, B^{\text{input}}_{\text{red}}, B^{\text{input}}_{\text{blue}}, B^{\text{input}}_{\text{pink}}, S^{\text{input}}, I^{\text{input}}, G^{\text{input}}]$, with the overall input being a 3D matrix of shape (temporal length n, context length $k = 7$, hidden size h), not counting the batch dimension. Seff *et al.* (2023) used a set transformer to learn a holistic representation of relative agent states, but set transformers are permutation invariant. In the problem that paper approached, the information had no need to be ordered since their model was predicting a horizon of future timesteps based on a uniform representation of relative states between different agents. In the case of LanSAR, the order of context information has some importance, with each index in the context sequence representing a different modality. To account for this difference, LanSAR takes inspiration from Devlin *et al.* (2019), in which the embedding for a [CLS] token is prepended to the sequence. Figure 3.2 provides additional visual clarity on the context encoder's input.

$$R = [\text{CLS}, X^{\text{input}}, B^{\text{input}}_{\text{red}}, B^{\text{input}}_{\text{blue}}, B^{\text{input}}_{\text{pink}}, S^{\text{input}}, I^{\text{input}}, G^{\text{input}}] \tag{3.13}$$

Positional encodings are then added to each value at the corresponding index in the new context sequence as a way to represent the different modalities.

$$P^{\text{context}} = [P^{\text{context}}_{\text{emb}}(i),\ i \in \{0, 1, ..., 7\}] \tag{3.14}$$

$$R = [r_i + p^{\text{context}}_i,\ i \in \{0, 1, ..., 7\}] \tag{3.15}$$

The output corresponding to the [CLS] token is taken as the encoding of the entire context sequence. The decision to follow this methodology was so the encoder could

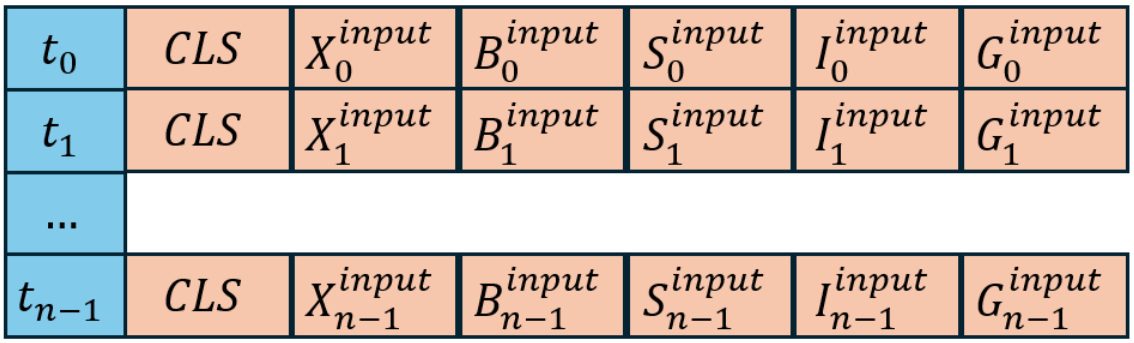


Figure 3.2: Context Sequence Visualization

directly learn a holistic representation of the sequence rather than forcing it to learn based on an arbitrary pooling of the output sequence.

$$E = \text{ENCODER}(R) \tag{3.16}$$

The context encoder follows a conventional multilayer encoder architecture as described in figure 2.1 and figure 2.2, with its final output being a sequence E, where the context dimension is condensed to a single value, resulting in a 2D matrix of shape (n, h), again not counting the batch dimension.

3.1.5 Motion Decoder

The motion decoder is intended to predict an immediate, one-step action based on both immediate and past environmental contexts and robot states, possessing memory of how the environment has changed over time to determine the next most optimal action. The output of the encoder is interleaved with the input projection of the natural language command C_{input}, resulting in the sequence

$$M = [C^{\text{input}}, E_0, C^{\text{input}}, E_1, ..., C^{\text{input}}, E_{n-1}] \tag{3.17}$$

which is used as the memory sequence for the decoder. Similarly, the target sequence is constructed by interleaving the target projection of the natural language command

with the target projection of the agent states.

$$T = [C^{\text{target}}, X_0^{\text{target}}, C^{\text{target}}, X_1^{\text{target}}, ..., C^{\text{target}}, X_{n-1}^{\text{target}}] \tag{3.18}$$

The command is interleaved to help reinforce the decoder in understanding the relation between the command and the robot motion. Other experiments that involved only prepending the command to the sequence resulted in deteriorated performance.

The memory and target sequence both have positional encodings added before being causally masked and sent through the decoder, which similarly follows the conventional multilayer decoder architecture as described in figure 2.1 and figure 2.3.

$$P^{\text{input}} = [P_{\text{emb}}^{\text{input}}(i),\ i \in \{0, 1, ..., (n-1) * 2\}] \tag{3.19}$$

$$M = [m_i + p_i^{\text{input}},\ i \in \{0, 1, ..., (n-1) * 2\}] \tag{3.20}$$

$$P^{\text{target}} = [P_{\text{emb}}^{\text{target}}(i),\ i \in \{0, 1, ..., (n-1) * 2\}] \tag{3.21}$$

$$T = [t_i + p_i^{\text{target}},\ i \in \{0, 1, ..., (n-1) * 2\}] \tag{3.22}$$

$$D = \text{DECODER}(T, M) \tag{3.23}$$

3.1.6 Output Layers

The values at odd indices of the decoder's output, which correspond to positional values in the interleaved sequence, are extracted

$$P = [d_i \mid i \text{ is odd}, i \in \{1, 2, ..., (n-1) * 2\}] \tag{3.24}$$

and sent through two separate MLP heads to decode the final outputs.

$$\Delta_p, \Delta_o = M_{\text{motion}}(P) \tag{3.25}$$

$$A = M_{\text{gripper}}(P) \tag{3.26}$$

Chapter 4

MATERIALS & METHODS

This chapter intends to provide details on the methodologies and resources used for both training and evaluating LanSAR. Information on the dataset and training routine are provided in detail, followed by information on the methods of evaluating model performance. Ablation experiments will be discussed to better understand the learning dynamics of LanSAR. Other approaches to this problem will then be discussed for the sake of comparison. Lastly, the methodology for examining the model's ability to robustly account for its own errors will be provided.

4.1 Dataset

The CALVIN dataset (Mees *et al.*, 2022b) consists of language-annotated, simulated demonstrations of robot actuator tasks. CALVIN was created specifically for the training and evaluation of long-horizon continuous motion tasks conditioned on language commands. The dataset was recorded from VR teleoperation in a variety of simulated environments, simply labeled, "A," "B," "C," and, "D," where users were told to interact with the environment without dropping any objects off the desk. Short horizons of this teleoperation data were taken and labeled with a language command based on the action performed by the teleoperator during that window. This data is otherwise unstructured such that the dataset is robust, covering as much of the state space as possible, starting from various points in time for the language tasks.

The CALVIN simulation environments are each different but maintain a similar structure. Each environment consists of a desk of a different color, texture, and

Figure 4.1: CALVIN Environment D

layout, but the same objects can be found in each environment. All environments contain a switch for toggling a light bulb, a button for toggling an LED light, a sliding door that moves to the left and right, and a drawer that can be opened and closed. The initial state of all interactable objects is random. All environments contain a blue, pink, and red block placed at a random but structured, accessible location. An example scenario for environment D can be seen in figure 4.1.

This dataset consists of multiple splits of data, but for the sake of this thesis, separate LanSAR models will be trained and validated on the "A,B,C $\rightarrow$ D" and "A,B,C,D $\rightarrow$ D" splits. The former contains 17,870 short-horizon, language-annotated demonstrations split across environments A, B, and C. The latter contains 22,994 demonstrations, now including environment D. Each demonstration consists of a sequence of at most 64 timesteps. The validation set for both data splits consists of 1,087 demonstrations on environment D. Outside of simply containing a different number of training samples, the primary difference in the datasets is that the former has a more difficult goal of generalizing to an unseen environment.

The dataset contains 34 categories of tasks, with each category having multiple examples that may have different language annotations for each, some of which are shown in table 4.1.

Task	**Example Commands**
lift_red_block_drawer	pick up the red block lying in the drawer; lift the red block lying in the drawer
turn_off_lightbulb	push the switch downwards; move down the switch
move_slider_left	slide the door to the left; move the sliding door to the left
open_drawer	open the drawer; grasp the drawer handle, then open it
turn_on_lightbulb	turn on the yellow lamp; toggle the light switch to turn on the yellow light
place_in_drawer	place the grasped object in the drawer; put the grasped object in the drawer
lift_pink_block_drawer	go towards the pink block in the drawer and pick it up; lift the pink block lying in the drawer
rotate_red_block_right	grasp the red block and turn it right; take the red block and rotate it right
rotate_pink_block_right	grasp the pink block and turn it right; grasp the pink block, then rotate it right
turn_off_led	turn off the green light; push down the button to turn off the green light
turn_on_led	toggle the button to turn on the led; push the button to turn on the green light
lift_red_block_slider	lift the red block lying on the shelf; lift the red block lying in the cabinet
place_in_slider	put the block in the slider; place the grasped object in the sliding cabinet
rotate_blue_block_left	rotate the blue block to the left; rotate left the blue block
push_pink_block_left	push left the pink block; push the pink block towards the left
unstack_block	remove the top block; take off the stacked block

lift_red_block_table	lift the red block; lift the red block from the table
move_slider_right	slide the door to the right side; grasp the door handle and slide the door to the right
lift_blue_block_table	grasp the blue block and lift it up; grasp the blue block on the table, then lift it up
lift_blue_block_slider	grasp the blue block lying in the cabinet; in the slider pick up the blue block
push_into_drawer	push the object into the drawer; sweep the block into the drawer
lift_pink_block_slider	grasp the pink block lying in the cabinet; in the cabinet grasp the pink block
lift_blue_block_drawer	grasp the blue block in the drawer; go towards the blue block in the drawer and lift it
stack_block	stack the object on top of another object; place the block on top of another block
rotate_red_block_left	take the red block and turn it left; grasp the red block and turn it left
close_drawer	close the drawer; grasp the drawer handle, then close it
push_blue_block_left	slide left the blue block; push the blue block towards the left
rotate_blue_block_right	take the blue block and rotate it right; rotate right the blue block
lift_pink_block_table	lift the pink block up; grasp the pink block on the table and lift it up
rotate_pink_block_left	take the pink block and rotate it left; take the pink block and turn it left
push_blue_block_right	go slide the blue block to the right; slide the blue block towards the right
push_pink_block_right	push the pink block to the right; go slide the pink block to the right
push_red_block_left	push the red block towards the left; slide the red block towards the left

push_red_block_right	sweep the red block to the right; slide the red block towards the right

Table 4.1: All Tasks Categories with Example Commands

4.2 Training

LanSAR is trained by first randomly initializing its parameters via Xavier uniform initialization (Glorot and Bengio, 2010), which can aid in preventing training failures via vanishing or exploding gradients, an issue in which gradients begin to either approach 0 or become extremely large, by keeping the variance of the inputs and outputs consistent. Each weight for a layer is sampled from a uniform distribution between a range calculated from the number of input and output features for that layer.

$$w_i^L \sim U(-a, a) \tag{4.1}$$

$$a = \sqrt{\frac{6}{\text{in}^L + \text{out}^L}} \tag{4.2}$$

The 22,994 example tasks from the A,B,C,D $\rightarrow$ D dataset are split into batches of 96, with padding added to allow a uniform maximum sequence length of 64, resulting in a total of 240 training batches. The 1,087 examples from the D split are utilized for validation. Due to the number of training batches, validation and training are desynchronized such that validation occurs every 25 steps of training rather than after a full epoch of training.

The loss function consists of a composite smooth L1 loss taken directly on motion deltas and a binary cross-entropy loss on gripper actions.

$$L_{\text{a}}^{\Delta} = \frac{1}{N} \sum_{i=0}^{N-1} \begin{cases} 0.5 \left(\Delta_{\text{a},i}^{\text{pred}} - \Delta_{\text{a},i}^{\text{true}} \right)^2 / \beta, & \text{if } \left| \Delta_{\text{a},i}^{\text{pred}} - \Delta_{\text{a},i}^{\text{true}} \right| < \beta \\ \left| \Delta_{\text{a},i}^{\text{pred}} - \Delta_{\text{a},i}^{\text{true}} \right| - 0.5\beta, & \text{otherwise} \end{cases} \tag{4.3}$$

$$L_{\mathrm{o}}^{\Delta} = \frac{1}{N}\sum_{i=0}^{N-1}\begin{cases} 0.5\left(\Delta_{\mathrm{o},i}^{\mathrm{pred}} - \Delta_{\mathrm{o},i}^{\mathrm{true}}\right)^2/\beta, & \text{if } \left|\Delta_{\mathrm{o},i}^{\mathrm{pred}} - \Delta_{\mathrm{o},i}^{\mathrm{true}}\right| < \beta \\ \left|\Delta_{\mathrm{o},i}^{\mathrm{pred}} - \Delta_{\mathrm{o},i}^{\mathrm{true}}\right| - 0.5\beta, & \text{otherwise} \end{cases} \tag{4.4}$$

$$L_a = -\frac{1}{N}\sum_{i=0}^{N-1}\left[A_i^{\mathrm{true}}\log(A_i^{\mathrm{pred}}) + (1 - A_i^{\mathrm{true}})\log(1 - A_i^{\mathrm{pred}})\right] \tag{4.5}$$

$$L = L_p^{\Delta} + L_o^{\Delta} + L_a \tag{4.6}$$

The choice to use smooth L1 loss instead of mean squared error (MSE) or root mean squared error (RMSE) is due to the latter two proving highly susceptible to orientation errors, easily spiraling out of control at inference time. Utilizing mean absolute error (MAE) for larger errors and MSE for smaller errors based on a threshold β resulted in a model capable of far more refined movements at the cost of failing to adequately learn rotation for certain tasks at higher thresholds of β. Lowering this threshold to behave closer to MAE did result in rotation being learned but also resulted in a decrease in motion delta accuracy. In the case of the best model, β was set to 0.1, which resulted in a significantly higher percentage of tasks being accomplished.

Linear decay was used for the learning rate scheduler, starting at a maximum learning rate of 4e-4 after a warmup of 5 epochs and decreasing to a minimum learning rate of 0. AdamW was used as the optimizer with a weight decay of 0.01. For additional regularization, a dropout rate of 0.1 was employed. Standard scaling was used for end effector position and orientation, robot joint states, drawer joint states, sliding door joint states, and end effector deltas. Thus, any single input feature x_i is scaled as

$$x_i^{\mathrm{scaled}} = \frac{x_i - \mu_i}{\sigma_i} \tag{4.7}$$

Standard scaling was selected in place of min-max scaling, in which values are scaled between a given range based on the minimum and maximum values of features in the dataset, as min-max scaling resulted in highly unstable training that failed to

Hyperparameter	**Value**
Transformer Hidden Size	512
Context Encoder Layers	6
Context Encoder Heads	16
Motion Decoder Layers	6
Motion Decoder Heads	6
Maximum Sequence Length	64
Dropout	0.1
Smooth L1 β	0.1
Batch Size	96
Optimizer	AdamW
Scheduler	Linear Decay
Starting Learning Rate	4e-4
Final Learning Rate	0
Weight Decay	0.01
Epochs	125

Table 4.2: Model Hyperparameters

learn any meaningful mapping from input to output. The same held true when no scaling was applied.

Table 4.2 provides a complete breakdown of all training and model hyperparameters.

4.3 Evaluation

4.3.1 CALVIN Challenge

Performance will be evaluated on the CALVIN Long-Horizon Multi-Task Language Control (LH-MTLC) challenge. This challenge involves testing a model in a variety of procedurally generated, simulated environments. The CALVIN simulation is built atop PyBullet (Coumans and Bai, 2016), a Python module for physics and robotics simulation. This environment utilizes environment D as the base environment, the same as with the data in the validation split. The simulator provides a Python interface for accessing all of the necessary data, including RGB static camera images, RGB gripper camera images, robot end effector position and orientation, robot joint states, and any other required information. The simulator requires a custom class and inference function to be written to interact with custom models. The simulation is otherwise set to utilize the default values of 360 timesteps per task with 1,000 total test sequences. Tests consist of a chain of tasks, in which the model has the aforementioned 360 timesteps to accomplish the current task before moving on to the next task. If the model fails to complete a task, all subsequent tasks in the chain are also considered to be failed. The commands and objects are generated such that no task in the chain should be impossible to complete. The overall goal of this challenge is to test the robustness of a model in not only handling unseen manipulation tasks but a series of them in a row, with each in the chain starting where the previous task was completed.

The inference function was written such that the required inputs described in section 3.1.3 and figure 3.1 are retrieved from the simulation and scaled to the expected input scale. The model tracks a window of history, appending new information up to a sequence length of 64, then incorporates a sliding window if the length exceeds this

amount, allowing the model to track a continuous history without extending beyond sequence lengths seen during training. Every 120 timesteps, the model is allowed to clear its history, which aids in the case where the model occasionally gets stuck in a loop of repetitious motions or if it gets "distracted" by a task misaligned with the language goal. Predicted deltas are scaled back up to the original state space, and the positional deltas and z-axis orientation delta are multiplied by an additional scaling factor. This positional delta scaling factor starts at 3 and is gradually reduced to 2 as the model's history sequence approaches a length of 64:

$$\text{scale} = 3 - \frac{\text{history}_{\text{current}}}{64} \tag{4.8}$$

The z-axis is scaled by a constant factor of 1.5. All deltas are then added to the previous state, giving the final action that's fed back to the simulation. This scaling factor was determined via hyperparameter tuning across multiple inference runs. This scaling proved necessary as the model otherwise often favored staying near the center of the state space or produced small motion deltas that would only take it near task completion.

Success rates per task in the task chain and individual task success rates will be tabulated and visualized to show the model's capability to perform a wide range of tasks across long horizons. Criteria for determining if a model completes a task are built into the CALVIN simulation, where checks are performed to see if the position, orientation, and joint states of the different objects have changed to match the anticipated end state of a desired task.

4.3.2 Ablation Studies

Ablation studies will be performed to further study the model's learning dynamics. Each ablation will be trained on the A,B,C,D $\rightarrow$ D split and tested on the CALVIN

LH-MTLC challenge, tabulating and comparing the results.

Prepended Language Encodings

In this experiment, rather than interleaving language commands, language commands will only be prepended to the beginning of the sequence. The expectation is that if the language modality is not repeated to reinforce it, the model may potentially learn to begin ignoring the command in favor of paying attention to environmental and robot state histories, in turn at least marginally decreasing performance.

Removal of Spatial Context

To study the effect non-image information had on the model's training, all information regarding agent states, block states, and hinge, light, and button states will be removed from the context sequence. It is anticipated that the model's performance will notably deteriorate with the loss of a richer environmental context, but not to a substantially impactful extent due to the robust information provided by both sets of image sequences and the sequences of robot states.

Removal of Image Context

To study the effect image information had on the model's training, all image encodings will be removed from the context sequence, forcing the model to rely exclusively on the end effector's position and global scene information. The expectation is that performance will deteriorate significantly for non-block interactions, given the lack of direct positional information provided for the drawer, door, and lights.

4.3.3 Self-corrective Capabilities

Directly quantifying LanSAR's self-corrective capabilities on a large scale is not feasible due to a wide range of possible edge cases to account for with any form of automatic detection. Instead, to simulate this behavior across all attempted tasks, three additional experiments are run on the LanSAR model trained with the A,B,C,D $\rightarrow$ D split. In each experiment, a random perturbation is sampled from a Gaussian distribution centered at the predicted positional delta with a standard deviation of the same predicted value multiplied by some scaling factor γ.

$$\Delta_p \sim \mathcal{N}\left(\Delta_p, (|\Delta_p| \cdot \gamma)\right) \tag{4.9}$$

In the first experiment, a perturbation sampled with a standard deviation of 5 times the predicted value is added every 50 timesteps. In the second experiment, the standard deviation of the perturbation distribution is reduced to 2.5 times the predicted value but added with an increased frequency of every 25 timestaps. In the final experiment, the standard deviation is further reduced to 0.5 times the predicted value but is added every 5 timesteps. All perturbations are calculated after predictions are scaled using equation 4.8. The expectation is for performance to remain highly consistent across each experiment despite the model being pulled in random directions away from its intended destination, demonstrating the ability of the model to adjust to both large but infrequent motion errors and small but frequent motion errors.

4.3.4 Comparisons

Comparisons will be made to other approaches to the CALVIN LH-MTLC challenge and robot motion. Primarily, the CALVIN baseline agent and two models considered to have achieved state-of-the-art performance on the CALVIN LH-MTLC challenge will be compared. Results from a LanSAR model trained on both the

A,B,C,D → D and A,B,C → D splits will be tabulated and compared to the results reported for each of these models in their respective papers.

MCIL

MCIL agents are the baseline agents proposed by Lynch and Sermanet (2021) and demonstrated on the CALVIN challenge by Mees *et al.* (2022b). MCIL leverages a sequence-to-sequence variational auto-encoder (Kingma and Welling, 2022) to learn a latent plan space. A plan proposal recognition network predicts a proposal distribution from perceptual information and the goal, while the recognition network predicts another distribution from only perceptual information. A plan is sampled from the recognition network's distribution, and a decoder predicts an action from the plan, perceptual information, and goal. Kullback-Leibler (KL) divergence loss was utilized between a plan proposal and recognition networks to bring the distributions closer.

HULC

Proposed by Mees *et al.* (2022a), HULC expands upon the MCIL architecture, with the most notable difference being the use of gripper camera images, instead of only static camera images, and the use of a transformer to encode sequence information as opposed to a recurrent neural network. It attempts to solve the issue of similar language encodings for semantically similar commands via a contrastive loss, maximizing the cosine similarity between the corresponding language features and sequence of visual features while minimizing it for unrelated sequences.

GR-1

GR-1 is a transformer-based architecture proposed by Wu *et al.* (2023), which possesses an architecture more similar to LanSAR than MCIL and HULC. GR-1

takes a CLIP language encoding, visual information encoded by a separate pre-trained vision transformer (ViT), and robot state (3D position, 3D orientation) as input. It predicts special [OBS] and [ACT] tokens for generating predictions of the next video frame and the next robot action. GR-1, similarly to LanSAR, interleaves its language encoding into the sequence. GR-1 is pre-trained with the EGO4D (Grauman *et al.*, 2022) dataset consisting of 3,500 hours of language-annotated video to predict the next frame of video before GR-1 is then fine-tuned on CALVIN motion data. The loss function during fine-tuning consists of a composite loss on the video predictions, motion deltas, and gripper actions.

Chapter 5

EXPERIMENT RESULTS

In this chapter, statistical results from each experiment outlined in chapter 4 are provided. A deeper discussion of these results is reserved for chapter 6.

5.1 CALVIN LH-MTLC Challenge Results

In this section, the results of the LH-MTLC challenge discussed in section 4.3.1 are provided. Table 5.1 shows the success rate of the best model on the task chains. It should be noted that the success rates are global percentages representing long-horizon task success rates, meaning a failure in a prerequisite task counts towards subsequent tasks, as previously described. The model locally succeeds at individual tasks with a success rate of 65.17%. A categorical breakdown of local success rates per attempted task can be seen in figure 5.1. Table 5.2 numerically breaks down the number of tasks attempted by the model for additional clarity. The model most prominently struggles with any tasks involving block interactions but strongly performs on tasks involving desk interactions, such as any task involving the door, drawer, and lights.

Task	**Success Rate** (%)
1	69.70
2	46.70
3	26.90
4	17.10
5	9.30

Table 5.1: Task Success Rates

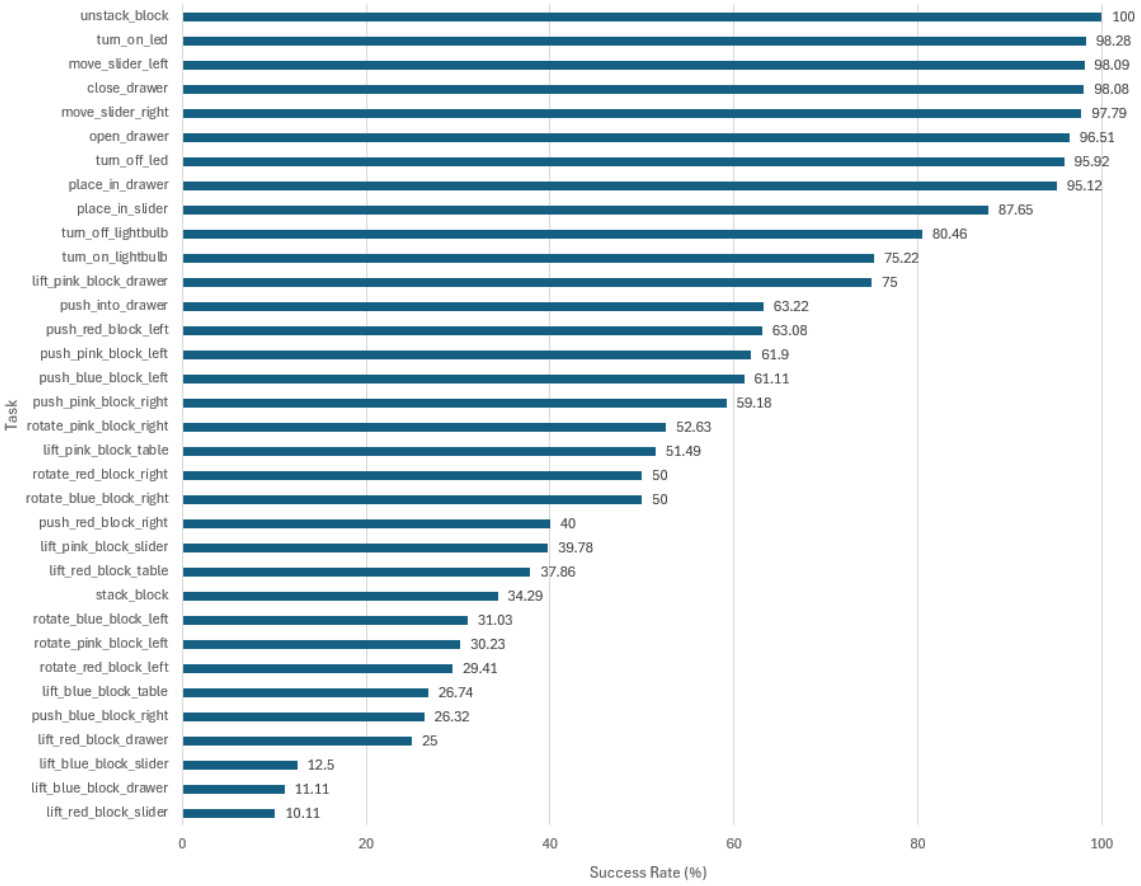


Figure 5.1: Categorical Task Success Rates

Task	Completions	Count	Success Rate (%)
unstack_block	7	7	100.00
turn_on_led	114	116	98.28
move_slider_left	154	157	98.09
close_drawer	102	104	98.08
move_slider_right	177	181	97.79
open_drawer	221	229	96.51
turn_off_led	94	98	95.92

place_in_drawer	39	41	95.12
place_in_slider	71	81	87.65
turn_off_lightbulb	70	87	80.46
turn_on_lightbulb	85	113	75.22
lift_pink_block_drawer	3	4	75.00
push_into_drawer	55	87	63.22
push_red_block_left	41	65	63.08
push_pink_block_left	39	63	61.90
push_blue_block_left	33	54	61.11
push_pink_block_right	29	49	59.18
rotate_pink_block_right	30	57	52.63
lift_pink_block_table	52	101	51.49
rotate_blue_block_right	30	60	50.00
rotate_red_block_right	31	62	50.00
push_red_block_right	24	60	40.00
lift_pink_block_slider	37	93	39.78
lift_red_block_table	39	103	37.86
stack_block	12	35	34.29
rotate_blue_block_left	18	58	31.03
rotate_pink_block_left	13	43	30.23
rotate_red_block_left	15	51	29.41
lift_blue_block_table	23	86	26.74
push_blue_block_right	15	57	26.32
lift_red_block_drawer	2	8	25.00

lift_blue_block_slider	12	96	12.50
lift_blue_block_drawer	1	9	11.11
lift_red_block_slider	9	89	10.11
Total	**1697**	**2604**	**65.17**

Table 5.2: Attempted Task Counts

5.2 Ablation Results

In this section, ablation study results as described in section 4.3.2 are provided. Results for the baseline LanSAR model and each ablation can be seen in figure 5.2 and table 5.3, which show results mostly matched what was anticipated. The prepended language command ablation experienced a notable decrease in performance in comparison to the baseline. Removal of spatial information resulted in a marginal decrease in performance, though not as large as anticipated. Removal of image information resulted in a substantial decrease in performance, but on different tasks than anticipated. When no image information is present, the model still performs well on tasks involving moving the sliding door to the left or interacting with the switch to enable or disable the lightbulb.

5.3 Self-correction Experiment Results

Self-corrective capabilities demonstrated by the perturbation experiment can be seen in table 5.4. Performance remains highly consistent, as anticipated, with the performance on corresponding tasks in the chain for each experiment being within 3% of one another, and all having a standard deviation of less than 1.

	Tasks Success Rates (%)				
Experiment	1	2	3	4	5
Baseline	69.70	46.70	26.90	17.10	9.30
Prepend	59.00	33.00	16.30	8.00	2.70
No Space	65.90	39.30	20.70	10.10	4.00
No Image	12.20	0.20	0.00	0.00	0.00

Table 5.3: Ablation Results

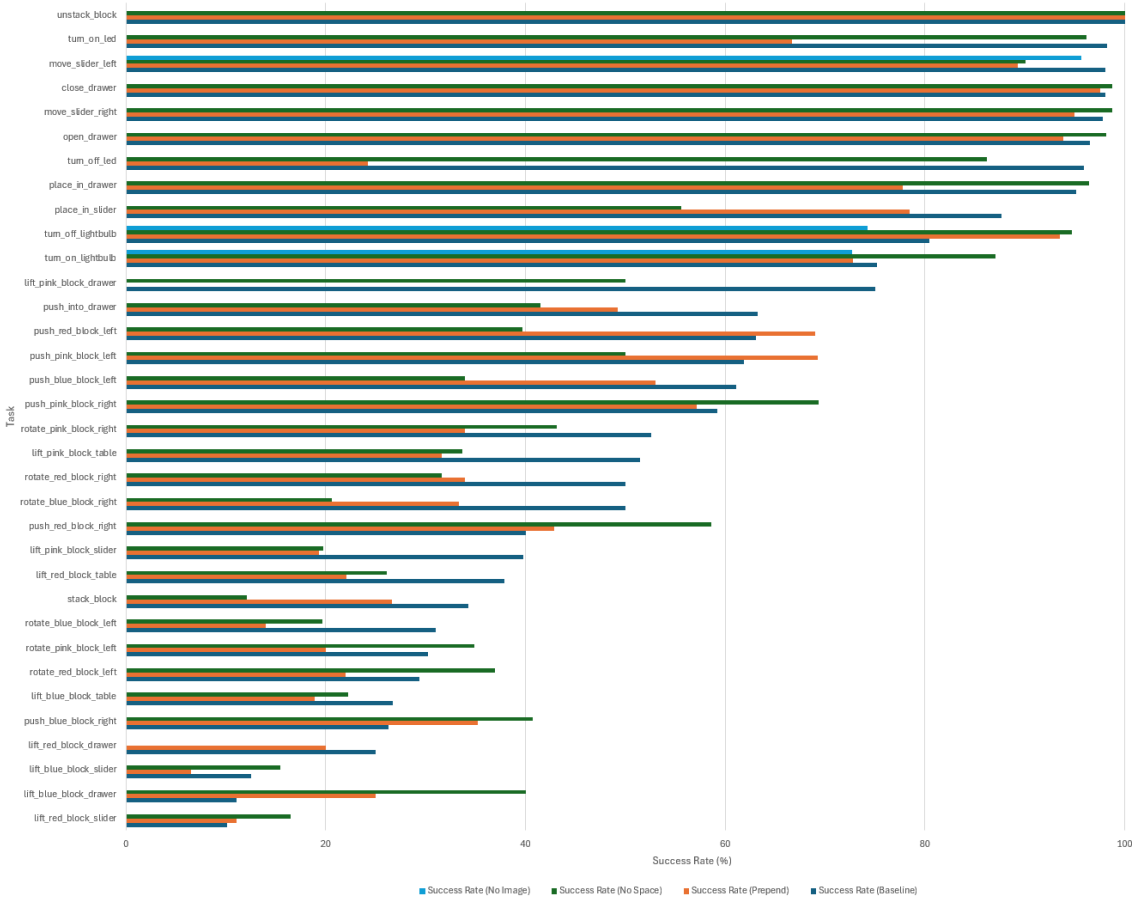


Figure 5.2: Ablation Categorical Task Success Rates

Perturbation Parameters		**Tasks Success Rates (%)**				
Scale	**Frequency**	1	2	3	4	5
0	0	69.70	46.70	26.90	17.10	9.30
5	50	70.30	45.80	27.80	16.00	7.90
2.5	25	71.00	46.70	28.00	16.20	9.50
0.5	5	70.50	47.80	29.50	17.60	7.80
Standard Deviation		0.4657	0.7089	0.9341	0.6534	0.7790

Table 5.4: Perturbation Results

5.4 Comparative Results

Figure 5.5 shows a comparison of two LanSAR models, one trained with all four simulation environments and one trained with the goal of generalizing to an unseen environment, to the baseline MCIL agent and the state-of-the-art GR-1 and HULC models. In both cases, LanSAR outperforms the baseline agent, but does not outperform any state-of-the-art models when trained with all four environments, only coming closest to HULC. LanSAR does, however, outperform HULC when generalizing to an unseen environment.

Experiment		**Task Success Rates (%)**				
Approach	**Data Split**	1	2	3	4	5
MCIL	ABCD→D	37.30	2.70	0.17	0.00	0.00
GR-1	ABCD→D	94.90	89.60	84.40	78.90	73.10
HULC	ABCD→D	88.90	73.30	58.70	47.50	38.30
LanSAR	ABCD→D	69.70	46.70	26.90	17.10	9.30
MCIL	ABC→D	30.40	1.30	0.17	0.00	0.00
GR-1	ABC→D	85.40	71.20	59.60	49.70	40.10
HULC	ABC→D	41.80	16.50	5.70	1.90	1.10
LanSAR	ABC→D	62.00	35.10	17.80	8.90	4.00

Table 5.5: Comparative Results to Other Approaches

Chapter 6

ANALYSIS & DISCUSSION

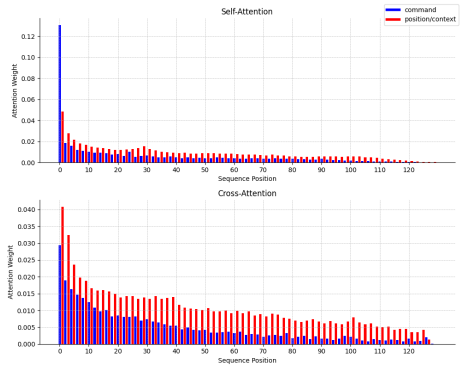
In this chapter, the results from chapter 5 will be discussed and tied together to provide a cohesive understanding of why LanSAR is performing at the level it does, and its strengths and weaknesses will be analyzed and explained. Additional examples of the model's behavior and the dynamics of its context attention mechanism will be provided. Avenues of future improvement will be then provided as a result of this analysis.

6.1 CALVIN LH-MTLC Challenge Analysis

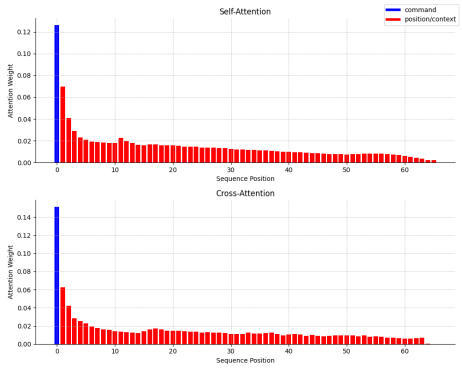
As previously shown, LanSAR accomplishes 65.17% of individually attempted tasks, with long-horizon success rates shown in table 5.1. Examining the performance on individual task types shown in figure 5.1 and table 5.2, there are clear groupings of performance capabilities for LanSAR. It proves performant on desk interactions but struggles on most block interactions, with three exceptions. The model shows a high success rate on storing blocks in the drawer and sliding door cabinet, as well as with unstacking blocks. In the former two cases, the model has already accomplished the more difficult prerequisite task of grasping a block, and these tasks fall more closely in line with a desk interaction. In the latter case, unstacking blocks can be easily performed accidentally by simply colliding with the blocks.

6.2 Ablation Analysis

As anticipated, prepending language commands resulted in deteriorated performance compared to the baseline LanSAR model. Investigating the temporal attention



(a) Interleaved command motion decoder attention



(b) Prepended command motion decoder attention

Figure 6.1: Comparison of Motion Decoder Attention Weights for Baseline & Prepended Language Commands

weights, shown in figure 6.1, for the initial timestep of a scenario given the command, "rotate the blue block to the right," a similar pattern of attention can be observed between the baseline and ablated models. Even in the interleaved approach, the initial command had the highest attention. However, the interleaved language commands are still attended to. Combined with the improved performance both as a whole and for most individual task types, this would indicate that interleaving the command does indeed reinforce the language modality, allowing the model to more strongly condition its actions based on it.

Examining the removal of spatial context, performance did deteriorate, but not as significantly as expected, showing that the model does somewhat rely on knowledge of global block and scene information but can still learn useful features with only knowledge of its own position and image data.

Further reinforcing the importance of even precomputed image encodings and showing spatial data primarily serves as supplementary information for this problem, the removal of image information drastically deteriorates performance. However, this

ablated model still accomplishes tasks involving moving the sliding door to the left or using the lever to interact with the lightbulb. Given the initial states of these objects are nearly identical for these tasks and the model saw environment D during training, this model effectively learned to memorize the motion required to complete these tasks, being otherwise unable to learn any form of meaningful environmental representation.

6.3 Self-correction Analysis

LanSAR is demonstrably capable of adjusting for previous mistakes and unfinished tasks to see them to completion. Table 5.4, as previously shown, qualitatively demonstrates this on a large scale by applying perturbations in both large, infrequent amounts and small, frequent amounts. To provide specific examples of self-corrective capabilities, if LanSAR is instructed to pick up and rotate a block, and then misses the block, it will not necessarily finish the associated "pick up and rotate" motion before trying again. Instead, it will often "realize" it does not possess the block and try again almost immediately. Similarly, if it does not open the drawer or sliding door all the way, it will also often "realize" this and move the gripper to the new location of the handle or inner wall of the drawer to complete the action. Figure 6.2 demonstrates the model's ability to self-correct when it initially fails to grab a red block. Figure 6.3 further demonstrates the model's ability to self-correct when the end effector's grasp slips on the sliding door handle halfway through the motion.

To examine this in further detail, figure 6.4 shows a 3D heatmap of the aggregated context attention weights over time for rotating the red block. Specifically, attention is aggregated to show which modalities are attended to the most per timestep, visualizing attention as the simulation progresses. The highest attention weights are on the image modalities, red block modality, [CLS] token, and end effector, in increasing

order. The image modalities provide constant information about the state of the environment, while the [CLS] token needs to represent the entire sequence, which would explain its high aggregated attention weight. The most important item of note is the high attention weights on the end effector and red block, which shows the model is, in particular, attending to the end effector and red block modalities, which would further explain the model's ability to understand when the block is and is not in the end effector's grasp. This also further reinforces the insights from the ablation experiment in which spatial context is removed, showing that it still contains useful information when paired with visual data.

Figure 6.5 show another heatmap of context attention weights over time for the self-corrected door slider. Again, the model pays high attention to the end effector position but has other notable spikes. The pink block is attended to more closely as the end effector passes near it before the scene modality is attended to as the end effector finishes manipulating the slider. The pink block attention still spikes far higher than the scene or even image attention, which highlights a problem that will be further discussed in section 6.5.

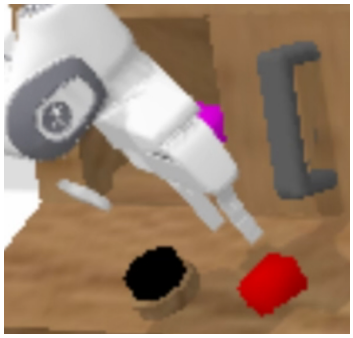
6.4 Comparative Analysis

6.4.1 MCIL

The differences between LanSAR and MCIL are significant, but the key differences are:

- MCIL predicts a latent plan as well as an action as opposed to only an action.
- MCIL uses its own vision network.
- MCIL acts more as a hybridized planner and reactive system, creating a latent

(a) Model misses red block



(b) Model stops lift-and-rotate action early

(c) Model tries again

(d) Model successfully grasps and rotates red block

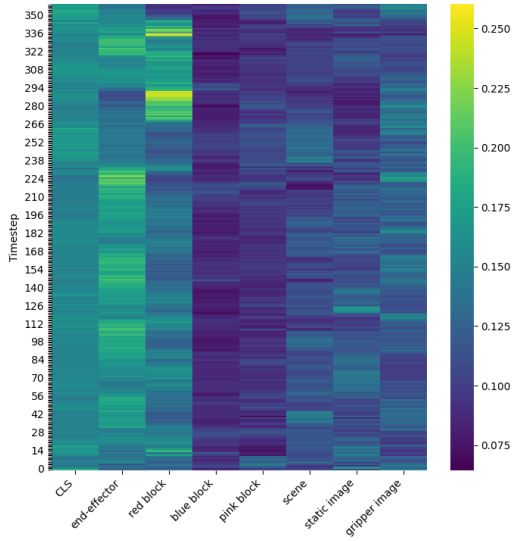
Figure 6.2: Example 1 of Model Self-Correcting

(a) Grasp slips on handle

(b) Model stops slide action

(c) Model tries again

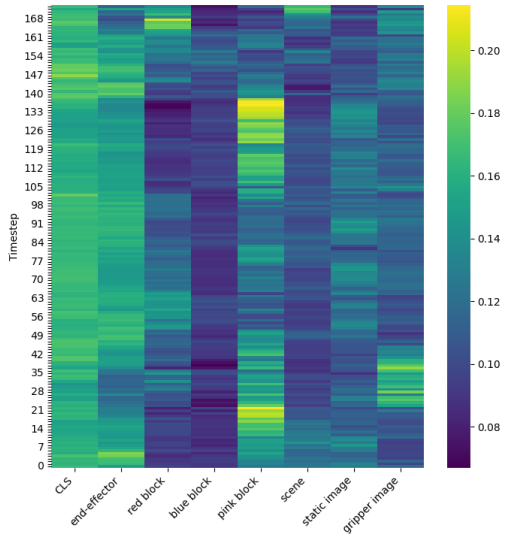
Figure 6.3: Example 2 of Model Self-Correcting



(a) Context attention weights

(b) Initial State

Figure 6.4: Context Attention Weights over Time for Self-Correcting Red Block Rotation



(a) Context attention weights

(b) Initial State

Figure 6.5: Context Attention Weights over Time for Self-Correcting Sliding Motion

plan and conditioning actions based both on it and immediate observations, and replanning arbitrarily.

- MCIL uses an RNN to process sequential information.

MCIL performs the weakest of all compared approaches, with LanSAR outperforming it by a considerable margin on both the A,B,C,D → D split and the A,B,C → D split. While MCIL has the advantage of its own vision network, it also uses an RNN for its sequential data processing, which can often struggle with long-term dependencies compared to transformer architectures.

6.4.2 HULC

The differences between HULC and LanSAR are much the same as MCIL, with a few additions:

- HULC again predicts a distribution as well as an action as opposed to only an action.
- HULC again uses its own vision network.
- HULC again acts more as a hybridized planner and reactive system.
- HULC utilizes a contrastive loss to better differentiate between similar language encodings.

HULC's use of a transformer for improved sequential data processing and contrastive loss to resolve the issue of similar encodings leads to substantially improved performance over MCIL and surpasses LanSAR's performance on the A,B,C,D → D split. However, HULC does not generalize well to unseen data, with LanSAR outperforming it on the A,B,C → D split. HULC's corresponding paper does not discuss

this lack of generalization in detail, but the contrastive learning mechanism could very well be overconstraining the model to training environments. As such, visual information from unseen environments could significantly deviate from the learned distributions in which corresponding commands and sequences should fall within.

6.4.3 GR-1

GR-1, being a transformer model that directly predicts motion from immediate perceptual information, is the closest approach to LanSAR, with the most important differences being:

- GR-1 is pre-trained on EGO4D to predict the next frame of video before being fine-tuned on CALVIN.
- GR-1 generates video frames as well as robot actions, whereas LanSAR only predicts robot actions.

GR-1's use of pre-training on a large motion dataset and then simultaneously fine-tuning for both video and motion predictions on the CALVIN dataset results in the highest performing model of all comparisons, outperforming MCIL, HULC, and LanSAR on all data splits.

In all cases, these models utilize only visual information and knowledge of the robot state, whereas the best LanSAR model includes spatial information. While LanSAR's ablation experiments and attention weight visualizations show an ability to correlate spatial and visual information despite being significantly differing modalities, it is indeed not realistic to assume that such knowledge would be available in a real-world scenario. Unless other detection models are incorporated to estimate the position of objects or the objects have some other methodology of being tracked, such as built-in GPS, LanSAR would only operate at its best in simulated applications.

Even with this limitation in mind, the performance of LanSAR with only visual information and knowledge of robot states still outperforms the MCIL baseline and generalizes to unseen environments more strongly than HULC. The most significant downside, in the case of LanSAR in comparison to MCIL, would be the quadratic memory requirement of transformers, which is not an issue with RNNs, albeit at the cost of reduced capability in handling long-term temporal dependencies. In comparison to HULC, LanSAR would be the ideal choice if the goal requires generalization to new environments. If the assumption can be made that the model will only operate in environments it was trained on and there is no risk of robot actuation errors and perturbations, then HULC becomes the ideal choice. It would be beneficial to perform further studies with HULC and other latent planning models to test their ability to correct for actuation errors while conditioning low-level actions on a latent plan, whereas LanSAR has already demonstrated consistent performance in such scenarios due to its predictions being based on the most immediate information available.

GR-1 proves itself to be the ideal choice in effectively all scenarios, substantially outperforming LanSAR and all other compared models. The only benefits LanSAR has in comparison would be a smaller model size of roughly 109 million parameters, including CLIP, as opposed to 195 million parameters, making it just under 56% of the size of GR-1. Additionally, LanSAR is trainable with a single-phase training routine with a single set of hyperparameters, as opposed to GR-1 requiring a pre-training phase.

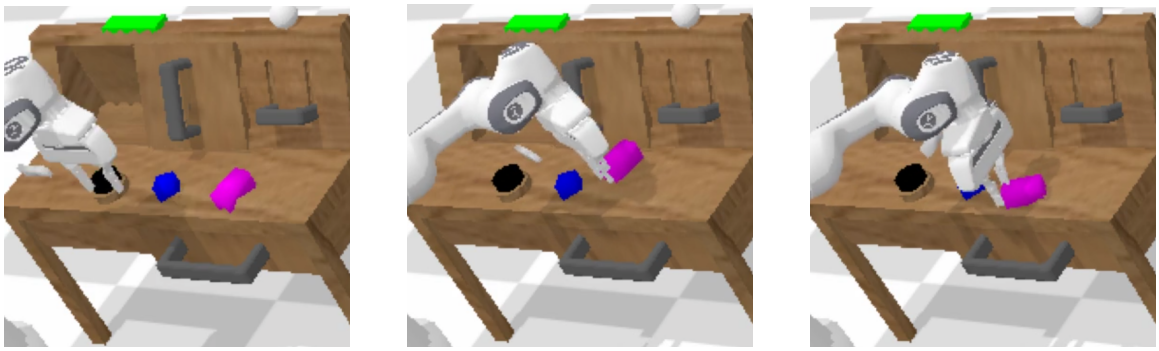
6.5 Understanding LanSAR's Shortcomings

Recall the breakdown of task success rates in figure 5.1 and table 5.2. LanSAR most notably struggled on tasks involving block interactions but performed well on desk interactions. Consider that there is a much wider variety of block states and

interactions with blocks as opposed to desk interactions. The language command and perceptual input for each demonstration should effectively act as an anchor, constraining the distribution of possible actions and separating the different tasks and demonstrations. Now again consider the issue described at the end of section 6.3 and shown in figure 6.5, in which the pink block is attended to for a task in which the model must open the sliding door. Despite image, scene, and end effector information being the most important information for this task, a block modality, irrelevant to this task, is still strongly attended to. Recall LanSAR's architecture as visualized in figure 3.1. The language modality is only considered by the motion decoder, making the context encoder language agnostic. In other words, the context encoder has the difficult task of learning a general representation of the environment for any possible downstream task and, for the same perceptual input, will always output the exact same holistic context as opposed to a task-specific context.

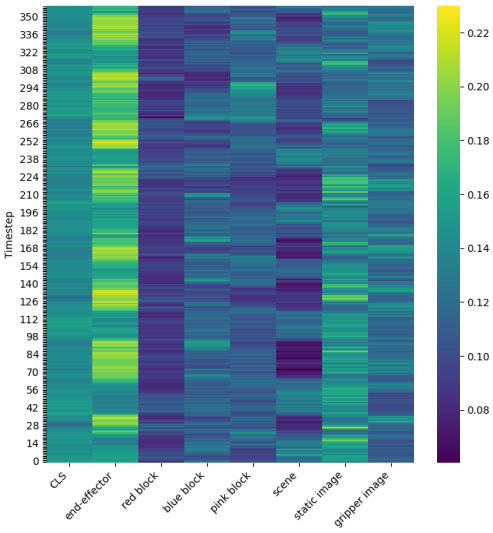
Figure 6.6 and 6.7 demonstrate this issue for an environment in which a blue and pink block are placed close to one another, with the model being instructed to pick up and rotate the blue block. The context encoder produces similar attention weights for both the blue and pink block, with small spikes for both at various points in time depending on the position of the end effector, showing it has indeed learned relevance based on spatial relationships, but has no understanding that the pink block is irrelevant to this scenario. Had it also been provided with the language command, it may have learned to attend more strongly to the blue block and less strongly to the pink block.

Figure 6.8 similarly demonstrates this issue for a more extreme failure case. In this environment, a red and blue block are placed next to each other, and the model is instructed to pick up the red block. For this experiment, the model does not produce any substantially meaningful motion, alternating between hovering around the red



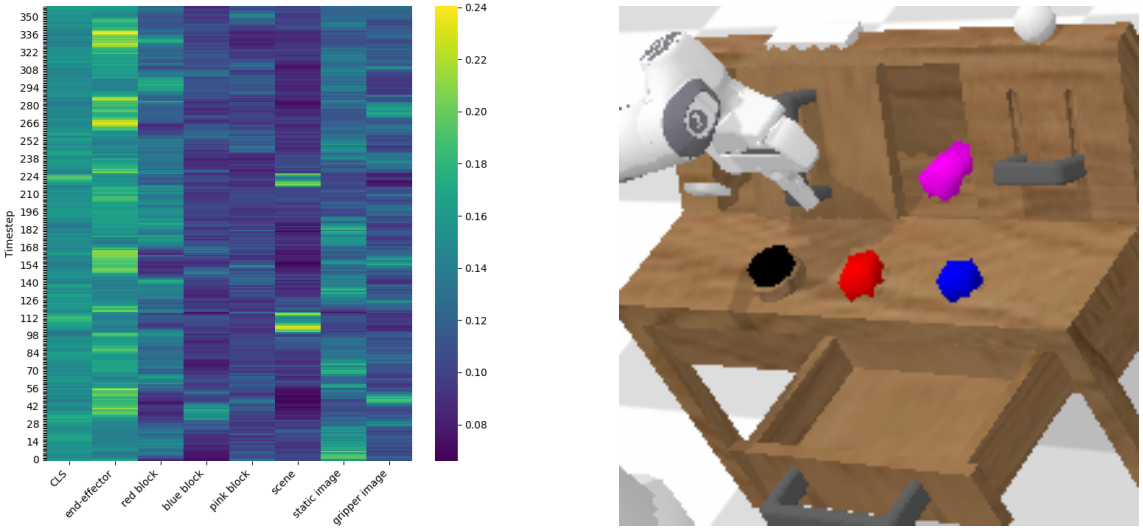
(a) Initial State (b) Grasps pink block (c) Rotates pink block

Figure 6.6: Misaligned Task Example: Instructed to Rotate Blue Block



(a) Context attention weights (b) Initial State

Figure 6.7: Context Attention Weights over Time for Misaligned Task



(a) Context attention weights

(b) Initial State

Figure 6.8: Context Attention Weights over Time for Motion Failure

block and blue block, occasionally grasping between or around the blocks with no success. The context encoder strongly attends to the end effector's position while increasing attention on the red and blue blocks depending on which block the arm is hovering closest to. There are also notable spikes in the scene attention when the arm accidentally pushes down on the LED button. This again shows the context encoder's ability to produce strong context encodings based on spatial relationships, but without any knowledge of the language command.

Now lastly consider the issue Mees *et al.* (2022a) highlighted of similar language command encodings and two language commands, "pick up and rotate the blue block," and, "pick up and rotate the pink block." Taking the CLIP encodings of these commands and computing the cosine similarity, a value of 94.29% is calculated. Effectively, LanSAR is failing to learn the specific semantics of such language commands and instead only understands a more general, "pick up and rotate the block," command.

When taking the language-agnostic nature of the context encoder and the similar language encodings seen by the motion decoder under consideration, it begins to become clear why LanSAR is struggling the way it does. Highly similar inputs are being provided for very different demonstrations, thus the inputs do not serve as adequate information to constrain the distributions of possible actions, thus the different tasks and demonstrations are not properly separated. In the best case, with no clear way of utilizing all input information to minimize the motion delta losses during training, the model can memorize the most common motion given similar environmental contexts and language commands, resulting in performing misaligned tasks such as seen in figures 6.6 and 6.7. In the worst case, the model may predict some weighted average of common motion, producing inaccurate or even nonsensical motion such as in figure 6.8.

6.5.1 Future Improvements

With these faults in mind and the strengths of the two state-of-the-art approaches under consideration, there are obvious avenues to improve LanSAR's performance. LanSAR's inability to properly separate different demonstrations due to its language-agnostic context encoder and highly similar language encodings is a substantial reason the model does not achieve state-of-the-art performance. While LanSAR does have the benefit of a simple training routine and is capable of learning from a small amount of data, 22,994 demonstrations form a considerably small dataset for training a transformer architecture. Many transformer architectures are trained with hundreds of thousands, if not millions or billions, of demonstrations. As such, pre-training with a larger amount of data would improve performance, as evidenced by Wu *et al.* (2023), where GR-1 was pre-trained with a dataset nearly 34 times larger than CALVIN. Taking all of this under consideration, the most immediately apparent ways to improve

the performance of LanSAR are as follows:

- Include language context as part of the context encoder's sequential input, allowing for a holistic, task-specific context encoding given the provided command and allowing the model to potentially learn to mask out irrelevant objects while providing more varied input to the motion decoder for each demonstration.
- Utilize contrastive learning techniques to further separate similar language commands, similar to HULC. It would also be beneficial to train or fine-tune a language model for LanSAR to learn new language encodings directly.
  - In this vein, training or fine-tuning a vision model instead of using precomputed encodings could also prove beneficial, but the CLIP image encodings proved robust to this problem. The contrastive learning mechanism could instead potentially be applied between the language command and the sequences of robot and context states without needing to directly bring the image encodings closer to the corresponding language encodings.
- Pre-train LanSAR on a larger motion dataset, similar to GR-1, and/or apply data augmentation to the CALVIN dataset.
  - Utilizing a sliding window to derive smaller sets of sequences from the existing sequence may prove beneficial, further leaning into the unstructured format of the CALVIN dataset. However, later positional encodings contain significant information about gripper actions, and this methodology could obfuscate that information.

LanSAR is additionally underfitting to the data with its current approach, as evidenced by figure 6.9, in which both the train and validation loss plateau at the same rate, with the model never overfitting to the training set. Resolving these issues by

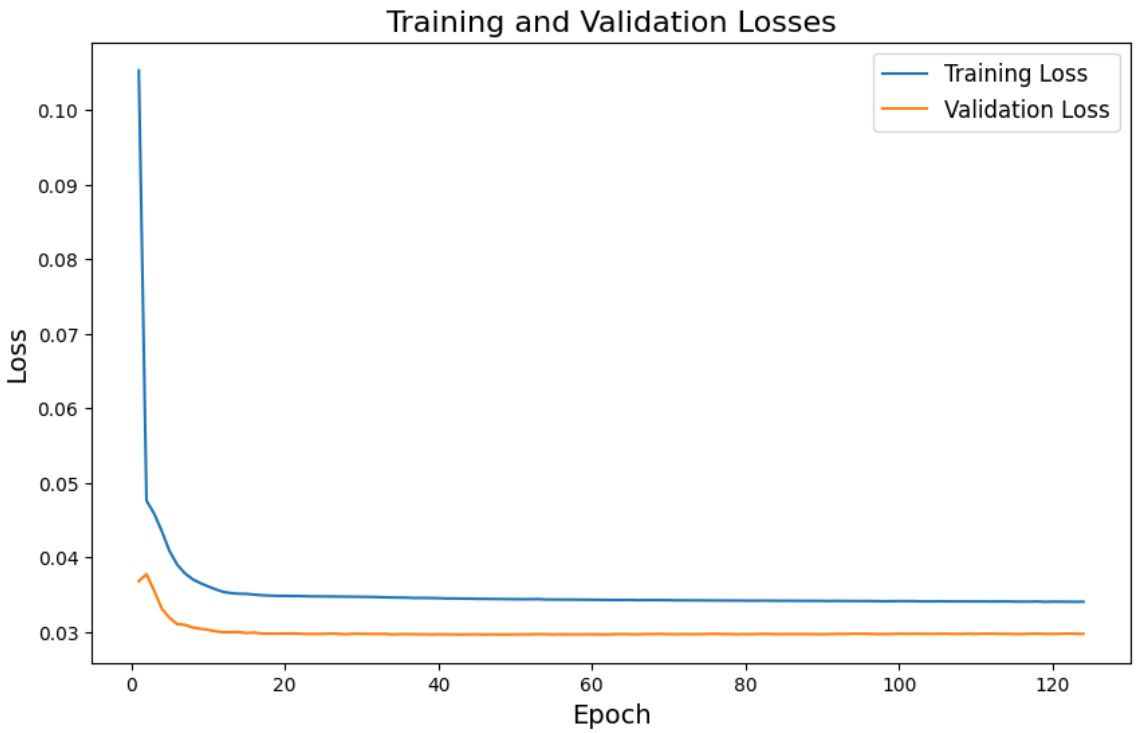


Figure 6.9: Best Model Loss Curves (MAE)

utilizing the additional proposed methodologies would provide a richer, more distinct representation of each demonstration, allowing the model to better understand the required actions for each task and better fit to the data. In turn, this would not only likely solve the issue of misaligned tasks but simultaneously resolve any remaining issues with accuracy of motion.

6.6 Conclusion

LanSAR was proposed as an approach for a real-time robot motion controller that incorporates a context encoder to encode multimodal visual and spatial data and a motion decoder that conditions on the context encodings, robot arm states, and language command to predict a motion delta. LanSAR is capable of leveraging a wide range of multimodal inputs to understand its environmental context, and can

perform language-commanded actions with a fairly high degree of accuracy. LanSAR was evaluated on the CALVIN LH-MTLC challenge, and while LanSAR does not achieve state-of-the-art performance, it does demonstrate an ability to correct its own errors in many scenarios and an ability to interact with the environment without any need for long-term planning.

LanSAR demonstrated capability in understanding visual and spatial relationships with its context encoder, which attended more strongly to different modalities depending on proximity to the end effector.

LanSAR particularly struggles with understanding the specifics of similarly encoded language commands, and its lack of task-specific context encodings can lead to similar or identical environmental encodings for different demonstrations. Both of these issues compound with one another, failing to adequately separate different demonstrations, resulting in LanSAR performing misaligned tasks or demonstrating a general loss in motion accuracy.

LanSAR may be improved by conditioning its context encodings on language commands rather than only in the motion decoding step. Additionally, utilizing contrastive learning techniques, such as the ones used in Radford *et al.* (2021) and Mees *et al.* (2022a) to separate similarly encoded language commands would prove beneficial and resolve the issue of LanSAR learning to simply memorize the most common motion for similar environment contexts and commands. Pre-training on a larger dataset, even if for a different domain such as video prediction, as is the case with the approach in Wu *et al.* (2023), would also prove beneficial by alleviating the data scarcity problem of the CALVIN dataset.

REFERENCES

Coumans, E. and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning", (2016).

Devlin, J., M.-W. Chang, K. Lee and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding", (2019).

Driess, D., F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch and P. Florence, "Palm-e: An embodied multimodal language model", (2023).

Glorot, X. and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in "Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics", edited by Y. W. Teh and M. Titterington, vol. 9 of *Proceedings of Machine Learning Research*, pp. 249–256 (PMLR, Chia Laguna Resort, Sardinia, Italy, 2010), URL `https://proceedings.mlr.press/v9/glorot10a.html`.

Grauman, K., A. Westbury, E. Byrne, Z. Chavis, A. Furnari, R. Girdhar, J. Hamburger, H. Jiang, M. Liu, X. Liu, M. Martin, T. Nagarajan, I. Radosavovic, S. K. Ramakrishnan, F. Ryan, J. Sharma, M. Wray, M. Xu, E. Z. Xu, C. Zhao, S. Bansal, D. Batra, V. Cartillier, S. Crane, T. Do, M. Doulaty, A. Erapalli, C. Feichtenhofer, A. Fragomeni, Q. Fu, A. Gebreselasie, C. Gonzalez, J. Hillis, X. Huang, Y. Huang, W. Jia, W. Khoo, J. Kolar, S. Kottur, A. Kumar, F. Landini, C. Li, Y. Li, Z. Li, K. Mangalam, R. Modhugu, J. Munro, T. Murrell, T. Nishiyasu, W. Price, P. R. Puentes, M. Ramazanova, L. Sari, K. Somasundaram, A. Southerland, Y. Sugano, R. Tao, M. Vo, Y. Wang, X. Wu, T. Yagi, Z. Zhao, Y. Zhu, P. Arbelaez, D. Crandall, D. Damen, G. M. Farinella, C. Fuegen, B. Ghanem, V. K. Ithapu, C. V. Jawahar, H. Joo, K. Kitani, H. Li, R. Newcombe, A. Oliva, H. S. Park, J. M. Rehg, Y. Sato, J. Shi, M. Z. Shou, A. Torralba, L. Torresani, M. Yan and J. Malik, "Ego4d: Around the world in 3,000 hours of egocentric video", (2022).

Ke, T.-W., N. Gkanatsios and K. Fragkiadaki, "3d diffuser actor: Policy diffusion with 3d scene representations", (2024).

Kingma, D. P. and M. Welling, "Auto-encoding variational bayes", (2022).

Lee, J., Y. Lee, J. Kim, A. R. Kosiorek, S. Choi and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks", (2019).

Lynch, C. and P. Sermanet, "Language conditioned imitation learning over unstructured data", (2021).

Mees, O., J. Borja-Diaz and W. Burgard, "Grounding language with visual affordances over unstructured data", (2023).

Mees, O., L. Hermann and W. Burgard, "What matters in language conditioned robotic imitation learning over unstructured data", (2022a).

Mees, O., L. Hermann, E. Rosete-Beas and W. Burgard, "Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks", (2022b).

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library", (2019).

Radford, A., J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger and I. Sutskever, "Learning transferable visual models from natural language supervision", (2021).

Seff, A., B. Cera, D. Chen, M. Ng, A. Zhou, N. Nayakanti, K. S. Refaat, R. Al-Rfou and B. Sapp, "Motionlm: Multi-agent motion forecasting as language modeling", (2023).

Vaswani, S., U. Parmar, G. Jones and P. Kaiser, "Attention is all you need", in "Advances in neural information processing systems", pp. 5998–6008 (2017).

Wu, H., Y. Jing, C. Cheang, G. Chen, J. Xu, X. Li, M. Liu, H. Li and T. Kong, "Unleashing large-scale video generative pre-training for visual robot manipulation", (2023).

Zhang, E., Y. Lu, W. Wang and A. Zhang, "Language control diffusion: Efficiently scaling through space, time, and tasks", (2024).

Zhou, H., Z. Bing, X. Yao, X. Su, C. Yang, K. Huang and A. Knoll, "Language-conditioned imitation learning with base skill priors under unstructured data", (2024).

Zhou, S., M. J. Phielipp, J. A. Sefair, S. I. Walker and H. B. Amor, "Clone swarms: Learning to predict and control multi-robot systems by imitation", in "2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)", (IEEE, 2019), URL http://dx.doi.org/10.1109/IROS40897.2019.8967824.

APPENDIX A

GITHUB REPOSITORY

LanSAR: https://github.com/anhardy/LanSAR