Dissertation

Transformer-based World Models in Cloth Manipulation

University of St Andrews

Faculty of Science
School of Computer Science

Florian Pfleiderer - 240030614

12th August 2025

Supervisor: Dr. Kasim Terzic

Conducted in partial fulfilment of the requirements for the degree of a
Master of Science (MSc) in Computer Science

Abstract

Robots are increasingly capable of manipulating everyday objects. Handling soft, flexible items such as cloth, however, remains a challenge. Tasks such as fabric flattening require predicting how cloth will deform in response to specific robot manipulations (e.g. pick-and-place actions) and using these predictions to plan subsequent actions. This thesis explores whether the Transformer, a modern neural network architecture, can improve prediction and planning in cloth manipulation, a domain not yet studied with this architecture. Earlier work on fabric flattening, including PlaNet-ClothPick, has used a type of *recurrent* model, the recurrent state-space model (RSSM), which processes information step-by-step over time while maintaining a memory of past states to learn cloth dynamics and guide planning. These methods often produce blurred predictions of future cloth configurations and require substantial training datasets, which limit their precision and efficiency. Building on recent successes of Transformer architectures in other domains, this thesis investigates whether replacing the recurrent model in PlaNet-ClothPick with a Transformer state-space model (TSSM) – capable of capturing long-range temporal dependencies – can overcome these limitations. This research tackles three challenges: (i) accurately modelling complex cloth dynamics, (ii) maintaining sharp spatial details over multi-step prediction horizons, and (iii) improving reward prediction for more effective planning in model-based reinforcement learning (MBRL). Results show the TSSM predicts sharper reconstructions, better preserves cloth boundaries, reduces reward prediction error, and improves planning performance relative to the RSSM baseline. This work provides an evaluation framework and establishes Transformers as a promising tool for deformable object manipulation.

Acknowledgements

First and foremost, I would like to extend my sincere gratitude to my thesis supervisor Kasim Terzic and to Abudureyimu Halite. Their unwavering support and guidance have been invaluable throughout my research and writing journey. I am truly thankful for their encouragement, which has fuelled my passion for the field of robotics and opened up new horizons for exploration.

Equally significant, I am deeply appreciative of the unwavering support from my family and girlfriend. Their encouragement has enabled me to stay focused and achieve the best possible results in my endeavours.

Declaration

I hereby certify that this dissertation, which is approximately *14800* words in length, has been composed by me, that it is a record of work carried out by me and has not been submitted in any previous application for a degree.

This project was conducted at the University of St Andrews between *05/2025* and *08/2025* in partial fulfilment of the requirements for the degree of *Master of Science in Computer Science*, under the supervision of *Kasim Terzic*.

In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web.

I retain the copyright of this work.

Signature:  **Date:** 12th August 2025

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

CDO cloth-like deformable object. 1–3, 5, 11, 13, 15, 16, 20, 26, 30, 31, 33–36, 39, 40, 42–44, 46–50

CEM cross-entropy method. 9, 11

FFN feedforward network. 23, 24

GRU gated recurrent unit. 17, 20, 23

LDM latent dynamics model. 1–5, 7–9, 11–16, 22–24, 27, 39, 48, 50

LSTM Long Short-Term Memory. 17

MBRL model-based reinforcement learning. ii, 1–5, 7–13, 15, 31, 33, 43, 48–50

MDP Markov decision process. 5–7, 9, 33

MHA multi-head attention. 23–25

MLP multi-layer perceptron. 20, 24, 40

MPC model-predictive control. 1–3, 9–12, 14, 33, 34, 39, 42–44, 46–49, 52

MSE mean squared error. 28, 39, 40, 48

NC normalised coverage. 42, 43, 46, 47

POMDP partially observable Markov decision process. 6

ReLU rectified linear unit. 19, 20, 24, 26

RL reinforcement rearning. 5–7, 9, 13, 15, 49

RNN recurrent neural network. 1, 2, 5, 8, 12, 14, 17, 22, 24, 47, 52

RSSM recurrent state-space model. ii, 1–5, 11, 12, 14, 16–18, 20–24, 26–28, 32, 33, 36, 40, 43, 47–50, 52

SotA state of the art. 1, 5, 14, 24

SSIM structured similarity index. 39, 48

TSSM Transformer state-space model. ii, 2–4, 12–14, 16, 19, 20, 22–24, 26–28, 30–37, 39, 40, 42, 43, 46–50

VAE variational autoencoder. 7, 8, 12

1. Introduction

Robot applications play an increasing role in everyday life. They already perform inspection, manufacturing, and assembly tasks effectively, but the most successful applications so far involve repetitive movements or structured, rigid-object manipulation. The next step in advancing robotic autonomy is to enable robots to operate independently in unknown and unstructured environments [1]. This would open the door for household robots to help with day-to-day tasks or assist in care roles. One major challenge that arises in such environments is the need to manipulate complicated, unstructured, and soft objects commonly referred to in the literature as cloth-like deformable objects (CDOs) [2]. In this thesis, I tackle this challenge by replacing the recurrent model used in prior cloth manipulation work with a Transformer-based architecture, the first time such a model has been applied in the CDO domain.

Manipulating CDOs is a long-standing and difficult problem in robotics. Their flexible nature, high degrees of freedom, complex non-linear dynamics, and the frequent occurrence of self-occlusion make them particularly challenging to model and control [3]. Automating tasks that involve CDOs, such as folding laundry, assisting with dressing, or handling fabrics in industrial settings, is crucial to developing robots that can offer meaningful assistance in unstructured real-world environments [1], [2]. Within this problem space, fabric flattening is considered a fundamental sub-task. Flattening a crumpled piece of fabric is a prerequisite for many downstream tasks such as folding. This is challenging because of the unpredictable way in which wrinkles, folds, and tension patterns change when parts of the cloth are grasped or moved [3].

One promising approach to solving such tasks is to let robots learn the dynamic behaviour of the cloth directly from data, enabling them to develop their own internal model of how different actions affect the state of the fabric. This is the central idea of model-based reinforcement learning (MBRL). In MBRL, an agent learns a model of the environment's dynamics from experience, and then uses this so-called latent dynamics model (LDM) to predict the state of the world as a result of an action (e.g. a force applied to the cloth) [4]. Planning algorithms use this to simulate (or imagine) a future state of the world to estimate how helpful a given action would be. For cloth flattening, a model can simulate how the cloth will move in response to a given pick-and-place action. A planning algorithm, such as model-predictive control (MPC), can then use these simulated rollouts to select the most promising next action [3].

In existing MBRL approaches to cloth manipulation, most state of the art (SotA) agents use a recurrent neural network (RNN) backbone to model temporal dependencies in a learnt LDM [5]–[8]. A well-known example is the recurrent state-space model (RSSM) [5], which maintains a latent state that encodes the history of past states and actions, updates it with each new observation, and predicts the subsequent latent representation from the current state–action pair. This architecture has been successfully applied in previous work, notably in PlaNet-ClothPick [9], a model-based framework for fabric flattening. RSSMs perform well in many domains, but face key limitations in CDO manipulation. Their sequential processing and limited memory make it difficult to capture complex, long-term dependencies and their imagined future states (priors) often appear blurry, indicating a limited latent state representation. In cloth flattening, these blurred predictions can lead to inaccurate pick points and planning failures [9]. Figure 1.1 illustrates this limitation in the RSSM used by PlaNet-ClothPick, as reported in previous work, and motivates

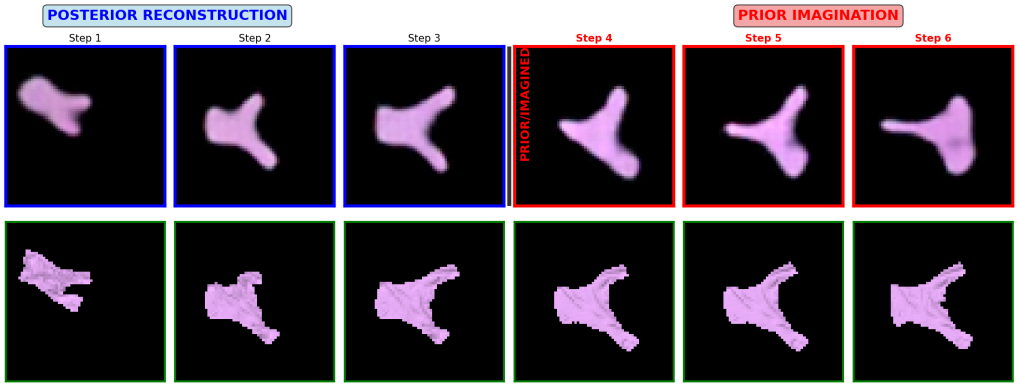


Figure 1.1.: Example of how the model (RSSM) reconstructs and predicts garment shapes over time. The bottom row shows the true garment configurations at each step. The top row shows what the model thinks the garment looks like. In the first three steps (blue boxes), the model is shown the real garment images (bottom row) and tries to recreate them – these are *posterior reconstructions*. In the last three steps (red boxes), the model is no longer shown the real images; instead, it predicts how the garment will move and change based only on its internal understanding of the dynamics and given pick-and-place actions – these are predictions (*prior imaginations*). The real images (bottom row) are only shown as a comparison reference. The predicted shapes are less accurate than posterior and ground truth, showing that the model struggles to keep fine details in memory.

the use of an alternative architecture capable of preserving fine spatial details in imagined states.

Motivation

This thesis introduces the first use of a Transformer-based latent dynamics model in the CDO domain, replacing the recurrent architecture used in prior cloth manipulation systems. Transformers have achieved superior performance to RNNs in various domains such as natural language processing, computer vision, and speech recognition[10]. The Transformer state-space model (TSSM)[11] is a Transformer-based LDM that allows the use of Transformers in MBRL, but has not yet been applied to cloth manipulation tasks. In this work, I adapt the TSSM architecture to replace the RSSM in PlaNet-ClothPick and integrate it into the existing evaluation framework (Agent-Arena) for direct performance comparison. The new model is trained on a garment manipulation dataset for fabric flattening tasks [9].

The motivation for replacing the recurrent model in PlaNet-ClothPick with a TSSM is rooted in the recent success of Transformer-based models and the two observed limitations of RSSMs. First, imagined priors generated during rollouts often lose fine detail, particularly at cloth contours. Because MPC samples pick points from these imagined states, any loss in spatial detail directly reduces planning quality [9]. Second, the RSSM is not data efficient: achieving competitive performance requires roughly an order of magnitude more training data than other MBRL systems in comparable domains. This suggests that RSSMs struggle to generalise the dynamics of the cloth from limited data [9].

The TSSM addresses both issues. Unlike RSSMs, which process sequences one step at a time, a TSSM uses self-attention to model all past states and actions directly [10]. This allows it to maintain a richer temporal context and to represent complex, long-range dependencies more effectively, which improves the quality of imagined rollouts and preserves

sharpness in reconstructions. Transformers have recently outperformed recurrent architectures in MBRL benchmarks such as Atari 100k [12], [13] and continuous control tasks [11], where integrating information across long horizons is essential. Cloth manipulation shares these requirements: the result of a grasp may depend on the full history of previous manipulations, the arrangement of wrinkles across the surface, and subtle changes in cloth tension. Furthermore, recent work suggests that TSSMs can achieve good data efficiency despite their larger parameter counts, especially when trained offline on large datasets [14], [15].

Research Goals and Novel Contributions

The goal of this work is to adapt a Transformer state-space model (TSSM) to the domain of cloth manipulation and investigate whether the Transformer-based architecture can outperform a recurrent state-space model (RSSM) in the context of cloth flattening under MBRL. To the best of my knowledge, this thesis presents the first application of a fully Transformer-based LDM to cloth manipulation under the MBRL framework.

In achieving the research goal, my contributions are as follows:

1. *Literature Synthesis* (Chapter 2): A systematic review of latent dynamics models for CDO manipulation, consolidating prior work in recurrent and Transformer-based architectures to identify gaps in cloth manipulation under MBRL.
2. *Theoretical framework* (Chapter 3): A detailed theoretical exposition and comparative analysis of the RSSM and TSSM, supported by novel figures illustrating their representational differences and the potential advantages of TSSM in modelling complex cloth dynamics.
3. *Architectural innovation* (Chapter 3-4): Redesign of a TSSM and integration into PlaNet-ClothPick framework, presenting the first Transformer-based LDM for CDO manipulation.
4. *System integration* (Chapter 4): The integration of the modified TSSM into the open-source Agent-Arena framework [16], enabling direct use and evaluation within various simulation environments for CDO manipulation.
5. *Comprehensive evaluation* (Chapter 5-6): Quantitative and qualitative comparison of TSSM and RSSM architectures on reconstruction quality, reward prediction accuracy, and closed-loop simulation planning performance in fabric flattening.

This thesis addresses the following research question: *Does a Transformer-based latent dynamics model improve latent state prediction and planning performance in cloth manipulation compared to a recurrent baseline?*

Summary of Results

Experiments show that the proposed TSSM produces sharper and more accurate reconstructions from imagined latent states than the RSSM baseline. Reward predictions from imagined states are also more accurate, improving the effectiveness of reward-based MPC in closed-loop simulation. This is, to the best of my knowledge, the first evaluation of a TSSM for CDO manipulation, and the results suggest that the strengths of Transformers in modelling long-term dependencies and preserving spatial detail translate into measurable gains for cloth manipulation tasks under MBRL. They also open the door for further analysis of the TSSM in the CDO domain.

Thesis Structure

The remainder of this thesis is organised as follows. Chapter 2 surveys the theoretical background and related work, including sequential decision processes, model-based reinforcement learning (MBRL), latent dynamics model (LDM), the Transformer architecture, as well as prior approaches to cloth manipulation. These form the conceptual foundation for my work. Chapter 3 presents a detailed theoretical exposition and comparative analysis of the RSSM and TSSM architectures, supported by novel figures illustrating their representational differences. Section 3.2.2 then details my modifications to the TSSM for cloth manipulation. Chapter 4 describes my implementation and integration of the adapted TSSM into PlaNet-ClothPick and Agent-Arena. Chapter 5 presents the experimental setup, metrics, and comparative results. Chapter 6 analyses the findings and discusses their implications. Finally, Chapter 7 concludes the thesis and summarises the main contributions.

2. Context Survey

This chapter reviews the theoretical foundations and existing literature relevant to deploying state of the art (SotA) model-based reinforcement learning (MBRL) techniques for the task of fabric flattening. In particular, it contextualises the use of latent dynamics models (LDMs) and explores recent architectural advances, most notably the replacement of recurrent state-space models (RSSMs) with Transformer-based dynamics, to address limitations in fabric manipulation tasks. Such tasks, including folding, flattening, and repositioning garments, involve interacting with cloth-like deformable objects (CDOs) [2], which have complex and non-linear dynamics. Learning a model of these dynamics from data to guide planning is a promising approach and commonly referred to as MBRL [4] and has achieved significant success in domains such as Atari, Go and Chess [7], [17], [18], but textile manipulation presents additional challenges. This thesis focuses on a canonical textile task, fabric flattening, as a benchmark to evaluate latent planning performance using Transformer-based architectures.

Throughout this thesis, I use the following terminology to describe components of MBRL systems. A *world model*, as defined by Ha *et al.* [19], acts as the internal imagination system of an agent, a neural network that learns to mimic a reinforcement rearning (RL) environment. It develops a simplified representation of how the environment operates, including how it changes and reacts to the agent's actions, allowing the agent to plan and make decisions without constant interaction with the real world. Within this broader world model framework, a *LDM*, is a core component responsible for predicting how the environment state will evolve in a compressed, lower dimensional *latent space*. Instead of predicting future images directly (pixel by pixel), it forecasts changes in this simplified underlying representation, from where the images can be reconstructed. The term *transition model* refers to the specific computational function within the LDM that takes the previous latent state and an action and then directly computes the next latent state [5]. The *backbone* (such as the recurrent neural network in RSSM) refers to the specific neural network architecture that implements the transition models functionality and handles the sequential processing and prediction of latent states [20]. A *rollout* is the process of simulating future steps given a list of actions, and the term *imagine* will be used interchangeably with *predict* and *simulate* to refer to this process.

To structure this chapter, the following components are covered: Section 2.1 illustrates how sequential decision-making processes can be formally modelled to produce a solvable problem. Section 2.2 covers how models can learn dynamics and plan actions in the framework of MBRL. Section 2.3 surveys Transformer-based approaches in MBRL and Section 2.4 covers advances and limitations of MBRL in fabric flattening, including a review of PlaNet-ClothPick and motivation for this thesis [9].

2.1. Sequential Decision Processes

Sequential decision processes provide a mathematical framework for describing how an agent interacts with its environment over time. A foundational model is the Markov decision process (MDP), which forms the basis for RL and MBRL algorithms. In a Markov process [21], the world is described as a set of possible states, where the next state depends only on the current one, this is the Markov property. Each state has possible rewards

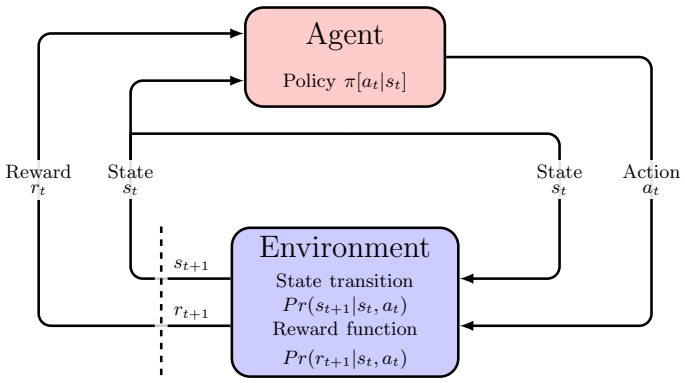


Figure 2.1.: Illustration of the reinforcement learning framework, depicting the cyclical interaction between the agent and the environment [22]. At each time step t, the agent observes the current state s_t and selects an action a_t according to its policy $\pi(a_t \mid s_t)$. The environment responds with the next state s_{t+1} and a reward r_{t+1}, determined by the transition probability $Pr(s_{t+1} \mid s_t, a_t)$ and reward function $Pr(r_{t+1} \mid s_t, a_t)$. This feedback loop continues iteratively, allowing the agent to learn an optimal policy over time.

$\mathcal{R}(r_{t+1} \mid s_t)$ and probabilities of moving to other states $\mathcal{T}(s_t \mid s_{t-1}, a_{t-1})$, depending on the agent's actions a_{t-1}. A Markov decision process extends a Markov process with a set of actions $\mathcal{A}$, assuming the state is fully observable. In practice, it is rarely possible to retrieve a full observation of the state, leading to partially observable Markov decision processs (POMDPs), where the agent receives observations $\mathcal{O}(o_t \mid s_t)$ that only partially reveal the state. A POMDP is defined by the tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, generating a sequence $(s_1, o_1, a_1, r_2), (s_2, o_2, a_2, r_3), (o_3, a_3, s_3, r_4), \ldots$ of states, observations, actions, transitions and rewards. The *discount rate* γ is used for reward accumulation (Equation 2.2). The policy π selects actions a_t based on the history $(o_{\leq t}, a_{<t})$, since s_t is unobserved and is formalised in Equation 2.1 [21].

$$\text{Policy:} \quad a_t \sim \pi(a_t \mid o_{\leq t}, a_{<t}) \tag{2.1}$$

This models many real-world scenarios with high complexity and partial state information [22]. This formal framework of a Markov process provides an organised way to describe the environment, an agent is acting in, and represents what the agent can explore by taking different actions, e.g. a robot moving around, making transitions between states. RL provides the sequential decision making framework to allow the robot (agent) to learn to act in an environment modelled by a POMDP, with the goal of maximising a reward that has to be defined [22]. This interaction is shown in Figure 2.1.

Reinforcement rearning (RL) [23] is a branch of machine learning (ML) where an agent learns through the interaction shown in Figure 2.1, either by learning a policy or value function through trial-and-error (model-free RL) or by building a model of the environment (model-based RL). Rewards, whether sparse or dense, signal the value of an action.

RL [23] extends the sequential decision-making framework to allow an agent to learn how to act in a POMDP-modelled environment. In each step, the agent observes the current state (or observation) s_t, selects an action a_t according to a policy $\pi(a_t \mid s_t)$, and receives the next observation s_{t+1} and reward r_{t+a} from the environment (Figure 2.1). The goal in RL is to maximise a cumulative return (reward) over time:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.2}$$

where $\gamma \in (0, 1]$ is the discount factor.

A value function estimates the expected return from a state under policy π:

$$V_\pi(\mathbf{s}) = \mathbb{E}_\pi \left(\sum_{k=0}^{\infty} \gamma^k \mathbf{r}_{t+k+1} \,\middle|\, \mathbf{s}_0 = \mathbf{s} \right), \tag{2.3}$$

where $\mathbb{E}$ is the expected future reward, given a policy π and subject to a discount rate γ.

and the optimal value function $V(\mathbf{s}) = \max_\pi V_\pi(s)$ satisfies the Bellman equation. This property is central to RL, as it enables the optimal policy to be obtained efficiently through dynamic programming or its approximations.

The core challenge in RL is to approximate the optimal policy or value function. This can be done directly through environment interaction in model-free RL, or indirectly in model-based RL by learning a model of the (often unknown) transition dynamics and rewards. Model-free methods do not use an explicit model of the environment and instead improve the policy directly from the experiences they collect. While conceptually simpler, they typically require many real interactions, leading to high sample complexity [24]. In contrast, model-based RL leverages a learnt or given model to simulate trajectories and plan ahead, reducing the dependence on costly trial-and-error [4].

A related setting is offline MBRL (or batch RL), where the model is trained entirely from a fixed dataset of past transitions, without further interaction with the real environment. By extracting as much information as possible from existing data, this approach enables safe and efficient learning in domains where online interaction is costly, impractical, or risky, such as safety-critical systems and resource-limited robotics [4], [25].

The key difficulty of this thesis lies in learning models that are both accurate and data efficient [25].

2.2. World Models in Reinforcement Learning

In many real-world tasks, the true MDP dynamics are unknown and must be approximated from data. In *deep* MBRL, neural networks learn this model from sensor inputs such as images or tabular data, enabling planning and policy learning without hand-crafted dynamics [26]. To improve data efficiency in MBRL, it is common to train a model of the environment in a compressed *latent space* rather than directly from high-dimensional observations. The resulting low-dimensional model, the *latent dynamics model (LDM)*, serves as an imitation of the environment for planning and policy learning. A complete world model includes both this latent model and an encoder–decoder (e.g., a variational autoencoder (VAE) [27]) that compresses and reconstructs the high-dimensional observations.

2.2.1. Architecture of World Models

World models are inspired by the manner in which humans are thought to construct a mental model of the world in which we live [25].

These models typically consist of three components: an encoder-decoder pair, a LDM, and a controller. The encoder-decoder pair is commonly implemented using a VAE [27]. A VAE encodes observations into a low-dimensional latent vector and reconstructs them back, training to minimise reconstruction error while preserving features that are most

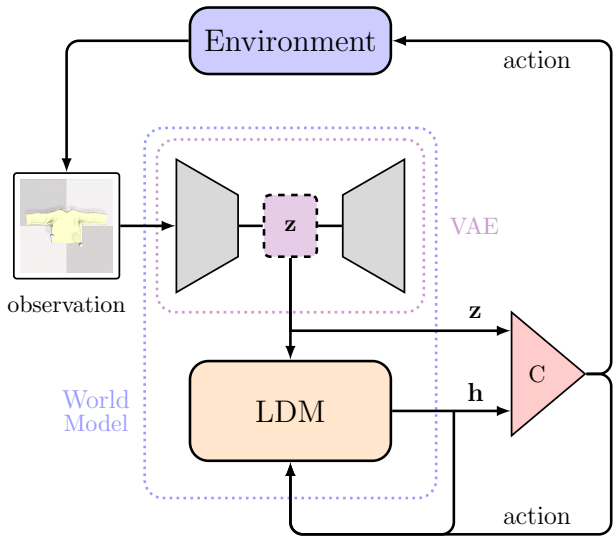


Figure 2.2.: General architecture of a world model used for end-to-end planning in model-based reinforcement learning [28]. This figure extends the interaction loop shown in Figure 2.1 by introducing a world model that replaces the environment with a predictive model operating in a compact latent space. The VAE compresses high-dimensional observations into latent variables z, which are processed by a latent dynamics model (LDM) to capture temporal dependencies. A controller (C) uses the stochastic latent state z and deterministic state h to select actions. This learnt model enables internal simulation and planning by predicting future states and rewards, forming a key component of modern MBRL pipelines. This figure was modified from Ha *et al.* [19].

relevant for downstream tasks such as control or planning [28]. The LDM uses an internal transition model to compute state transitions in the latent space, which is implemented using a recurrent neural network (RNN) [29], a structured state-space model (S4) [20], or a Transformer [10] as the backbone. The choice of backbone affects the model's ability to capture long-term dependencies and impacts both predictive accuracy and computational efficiency [20]. Finally, the controller selects actions based on the current latent state and hidden dynamics. Figure 2.2 illustrates the typical structure of such a world model: a VAE encodes observations into latent variables, which are processed by an RNN-based LDM to predict future states, and a controller (C) determines the next action based on the latent state and internal memory.

RNNs are a class of neural networks specifically designed to process sequences of data, making them excellent for tasks involving temporal dependencies. Unlike traditional networks, RNNs have an internal *memory*, which allows them to use information from previous steps to predict future states. They have been the predominant backbone architecture for world models, acting as a predictive model for how latent variables will evolve over time. However, traditional RNNs can struggle with long-term prediction due to limited memory capacity and vanishing gradients. To address these issues, alternative architectures like Transformers and S4 models are being explored [20].

2.2.2. Challenges in Learning Robust LDMs

Robust world models are the key to efficient and generalisable MBRL, enabling more sample-efficient planning and decision-making [20]. A central challenge is that LDMs suffer from error accumulation in multi-step predictions, which can degrade planning performance [5]. Strategies such as multi-step loss objectives or specialised n-step dynamics

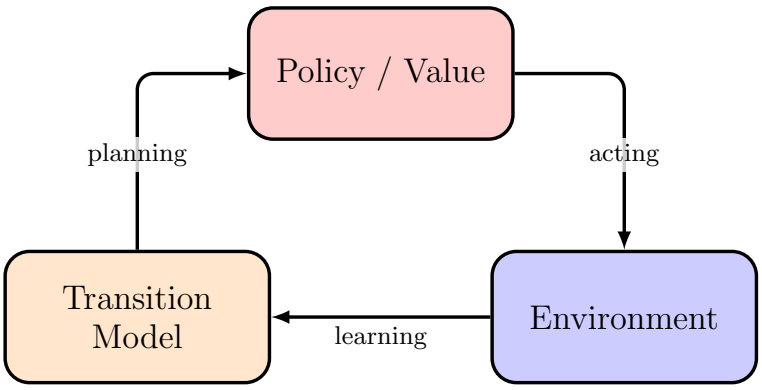


Figure 2.3.: This figure illustrates the iterative interaction between three key components: the agent's learnt transition model, its policy or value function, and the real environment. This loop complements Fig.2.2 and enables the agent to reason over imagined futures, especially under uncertainty and partial observability[25]. The illustration was modified from Moerland *et al.* [4].

models aim to reduce this compounding error [4]. Since methods like trajectory sampling, model-predictive control (MPC), and policy optimisation rely on accurate rollouts [6], [25], maintaining robustness beyond single-step accuracy is essential. In complex domains such as textile manipulation, real-world dynamics are rarely known exactly. MBRL agents must iteratively alternate between collecting new interaction data to refine the model (learning) and using the model to plan actions. This continual model–planning loop, shown in Figure 2.3, is especially important under uncertainty, stochastic transitions, and partial observability [4]. In a stochastic MDP, the transition function defines a distribution over possible next states rather than a single outcome. This stochasticity is a fundamental noise in the transition of the ground truth state that cannot be mitigated, making accurate long-term prediction more difficult and deterministic models may collapse, as they try to predict only the exact next state [4]. Partial observability adds further difficulty, as the agent must infer hidden aspects of the state from past observations. Recurrent architectures are commonly used to maintain a latent belief state over time, providing a compact summary of history that supports planning in high-dimensional sequential tasks [4], [28].

2.2.3. Latent Planning Methods and Models

The Dyna architecture [30] was an early model-based RL framework, that integrates learning, planning, and acting in a feedback loop (Fig.2.4), allowing agents to update policies from both real and model-generated experience. Modern MBRL methods such as PlaNet and Dreamer extend this structure by using deep neural networks for the learnt model[4]. Table 2.1 summarises common learning and planning approaches, highlighting *latent models* and *mpc*, which are central to this thesis. Although learnt LDMs can represent the dynamics of the underlying environment, a key challenge remains: How should the agent decide which action to take next?

Plaat *et al.* [25] propose a taxonomy (Table 2.1) that provides a lens through which to view the learning and planning components specifically within the MBRL context. I have added the differentiation *Online* and *Policy learning* to *Trajectory rollouts*, as this definition was originally to broad and is used by most models in this context survey. One common latent planning strategy is model-predictive control (MPC) [31], which uses a learnt dynamics model to simulate future state-action trajectories over a finite horizon. At each step, MPC searches for the sequence that maximises a reward (or minimises a cost), often using sampling-based optimisers such as the cross-entropy method (CEM) [5]. Only the first action from the optimal sequence is executed before the horizon is shifted

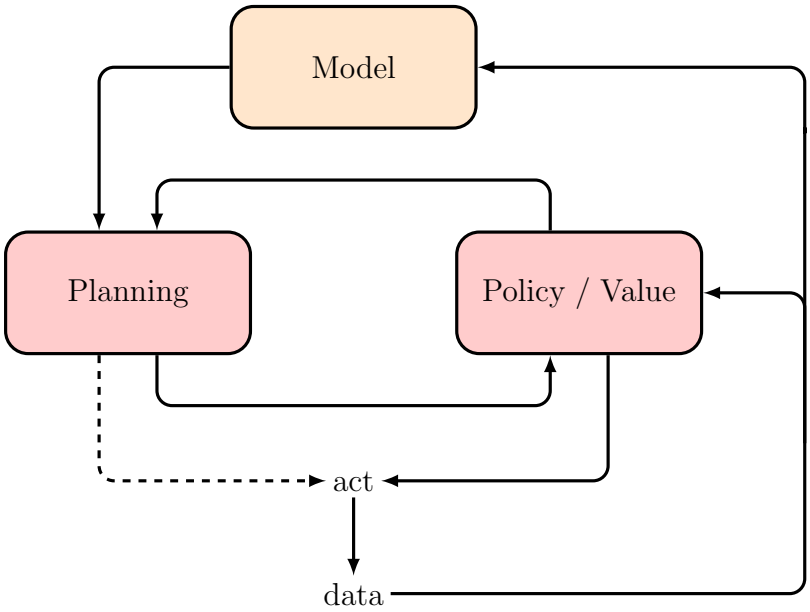


Figure 2.4.: The Dyna architecture [30] illustrates the foundational idea behind MBRL: integrating learning, planning, and acting in a unified framework. This feedback loop – combining environment interaction and model-generated trajectories – establishes a core template still used in modern MBRL methods. The dashed line represents the dataflow that many modern planning-based models like PlaNet follow, as they skip the Policy/Value networks, an extension to Figure 2.3.

Table 2.1.: Types of learning and planning commonly found in high-accuracy MBRL algorithms. Latent models and MPC are highlighted, as they will be the focus of this thesis. *Online planning* directly informs action selection, while *policy learning* trains a separate policy on these imagined trajectories offline. Modified from [25].

Learning	Planning
Probabilistic inference	Trajectory rollouts (Online / Policy Learning)
Ensemble methods	**MPC**
Latent models	End-to-end learning and planning

Table 2.2.: Planning algorithms of methods using LDMs as learning strategy.

Model	Planning	Action Space
DREAMER [6]–[8]	Trajectory rollouts	continuous / discrete
PLAN2EXPLORE [32]	Trajectory rollouts	continuous
PLANET [5]	MPC	continuous
SIMPLE [18]	Trajectory rollouts	discrete

forward and the plan recomputed with new observations. This receding horizon approach enables MPC to adapt to model errors and environmental changes, making it well suited for high-dimensional, uncertain domains. DREAMER uses what Plaat *et al.* [25] categorise *Trajectory Rollouts*, and I further define as *Policy learning*, where the imagined rollouts (the predictions) are used to train separate policy and value networks [6], making it an actor-critic MBRL algorithm, and many algorithms extend the DREAMER architecture.

Table 2.2 summarises key LDM approaches, excluding Transformer-based architectures and algorithms for CDO, discussed in Sec.2.3 and 2.4. SIMPLE[18] models discrete-action Atari games via video prediction, with policies trained inside the learnt model. PLANET[5] plans directly in latent space using online MPC with CEM, while DREAMER[6] replaces online planning with a latent actor–critic, improving efficiency through imagined rollouts. PLAN2EXPLORE[32] extends PLANET with exploration-driven action selection based on ensemble disagreement. Among these, PLANET's MPC and Dreamer's actor–critic are most relevant here, forming the basis of PLANET-CLOTHPICK for CDO manipulation [3] and TRANSDREAMER [11], respectively.

While this thesis focuses on model-based approaches, it is worth noting that model-free methods can also benefit from learning LDMs. Lee *et al.* [33] implement the actor-critic algorithm, where the agent learns a dynamic model to simulate the policy function (Equation 2.1), rather than the environment dynamics. This dynamic policy model now directly predicts actions given a state.

2.2.4. The Recurrent State-Space Model

Hafner *et al.* [5] proposed the recurrent state-space model (RSSM), a more robust MBRL framework for robotics. It combines deterministic and stochastic components to capture both long-term structure and uncertainty in environment transitions. and consists of a transition model (state model), an observation decoder, an encoder, and a reward predictor, all implemented by separate neural networks.

$$\begin{aligned} \text{Deterministic state model:} \quad & h_t = f\big(h_{t-1},\, z_{t-1},\, a_{t-1}\big) \\ \text{Stochastic state model:} \quad & z_t \sim p\big(z_t \mid h_t\big) \\ \text{Observation model:} \quad & o_t \sim p\big(o_t \mid h_t,\, z_t\big) \\ \text{Reward model:} \quad & r_t \sim p\big(r_t \mid h_t,\, z_t\big)\,, \end{aligned} \tag{2.4}$$

These equations formalise how RSSM evolves latent states. The hidden deterministic state h_t represents a long-term temporal structure, while the stochastic state z_t allows the model to represent uncertainty, crucial for exploration and robust prediction. The decoder and reward model decode these latent states into actionable information for learning and planning, while the encoder produces a latent representation of each observation [5].

RSSM was first applied in PLANET [5], which demonstrated its capacity for solving continuous control tasks by planning directly in the latent space using MPC. DREAMER[6]

replaced online planning with a learnt actor–critic policy in the latent space, improving efficiency and scalability. DreamerV2[7] showed that RSSM-based architectures also handle discrete actions, underscoring their versatility. The RSSM is a fundamental component of PlaNet-ClothPick used and discussed in this thesis, and Section 3.1 explains the architecture in detail.

2.3. Transformer Architectures in Latent Dynamic Models

Transformers address key limitations of RNNs, such as vanishing gradients and limited long-term memory, by replacing recurrent connections with attention mechanisms that operate over entire sequences in parallel. This allows direct modelling of long-range dependencies and efficient training [10], [20].

2.3.1. Transformer Architecture

The Transformer architecture relies entirely on so-called *attention mechanisms*, which it uses to compute what parts of the input sequence to *attend* to and what to ignore [10]. It follows a multi-layer encoder–decoder structure: the encoder processes the input through layers that apply attention and simple feed-forward transformations, while the decoder uses a similar setup but also ensures it cannot use information from future positions. At its core, attention determines how strongly each element in the sequence is influenced by others, and combines this information to produce the output representations used in the next layers [10]. The Transformer state-space model (TSSM) in this thesis uses an encoder-type Transformer, described in Section 3.2.1.

Transformers outperform RNNs in domains with long and complex input sequences [11]. Unlike RNNs, which rely on a compressed recurrent state, Transformers have direct access to all previous outputs, enabling the modelling of complex dependencies and parallel training [10], [20]. Their main drawback is quadratic time and space complexity when attending to all tokens, which limits long-range memory access. This has been mitigated through approaches such as adding recurrent elements [34] or caching methods for inference [15].

2.3.2. Transformer-based Models

In MBRL, Transformers are increasingly used as world-model backbones to learn accurate world models and temporally consistent simulated rollouts for policy learning [13], [20]. Table 2.3 summarises recent methods. All models use *trajectory rollouts* to learn either actor and critic models (*Policy Learning*) or other prediction mechanism (*Online Planning*), as opposed to using MPC for planning.

Iris[14] and Storm[12] encode observations into discrete VAE tokens for efficient sequence modelling, with Storm adding stochastic latent variables to improve rollout robustness. TransDreamer[11] replaces the RSSM in Dreamer with a Transformer to capture more complex temporal dependencies. Twister[13] uses an action-conditioned contrastive predictive coding (AC-CPC) to improve long-horizon planning, while Twm[15] employs a Transformer-XL for latent trajectory modelling but decouples the policy at inference for model-free speed. These variations illustrate how Transformer architectures can augment LDM planning through richer temporal modelling, uncertainty handling, or efficiency gains.

Table 2.3.: Planning algorithms of methods using Transformer-based LDMs as learning strategy. All models employ *trajectory rollouts* (imagination) as the broad planning category.

Model	Planning	Action Space
TransDreamer [11]	Online Planning	continuous / discrete
Iris [14]	Online Planning	discrete
Twm [15]	Policy Learning	discrete
Storm [12]	Policy Learning	discrete
Twister [13]	Policy Learning	discrete

Most Transformer-based MBRL methods are benchmarked on the Atari100k benchmark [35], and CDO manipulation is not a primary reported application area. Only TransDreamer is evaluated on a continuous control benchmark that resembles the skills needed for robotic manipulation and the domain of CDO manipulation [11].

2.4. Cloth Manipulation and Fabric Flattening

This section introduces the main problem domain of this thesis: robotic manipulation of cloth-like deformable objects (CDOs), in particular fabric flattening. Compared to rigid objects, CDOs exhibit high degrees of freedom, non-linear dynamics, and frequent self-occlusions [2], [3]. These properties make it difficult to learn accurate latent dynamics models for model-based RL, while error accumulation further complicates long-horizon planning [25]. Data-driven model-based methods, including offline learning, offer a promising balance of sample efficiency and generalisation across tasks, but remain difficult to apply to cloth flattening due to the need for large-scale data collection and robust simulation environments such as SoftGym [36]. This thesis addresses these challenges by implementing and evaluating a TSSM for fabric flattening.

2.4.1. Challenges in CDO

The manipulation of CDO remains difficult due to their complex, unpredictable dynamics and the inability of current simulators to fully replicate real physical behaviour, leading to reduced performance when transferring models from simulation to reality (sim-to-real gap) [2], [3]. Describing the exact state of a crumpled cloth is non-trivial, hindering accurate state representation, reward function design, and automated goal evaluation [37]. These issues lead to grasping performance, where robots often misidentify grasp points or pick incorrect layers. Moreover, deep RL and imitation learning methods demand extensive and diverse datasets, which are difficult to obtain for varied textile properties and tasks [3]. Progress is further limited by the lack of standardised benchmarks and real-world datasets with reliable ground-truth labels [2], [25].

2.4.2. Common Approaches

Approaches in the CDO domain vary from heuristic pipelines [38], model-free policy learning [39], to various forms of model-based control[9], [37], [40]–[43]. Model-based methods are either analytical (physics-based) or data-driven. Data-driven RL methods are particularly relevant and learn cloth dynamics directly from exposure to data, which can be represented as raw images, 3D point clouds, or latent representations [2]. Implementing a

Table 2.4.: Overview of approaches for cloth manipulation, highlighting learning methods and planning strategies.

Framework	Learning	Planning
PlaNet-ClothPick[9]	RSSM	reward-based MPC
CFM [40]	Contrastive Estimation	goal-conditioned MPC
VCD [41]	GNN-based LDM	reward-based MPC
VSF [43]	visual dynamics models	goal-conditioned MPC
G-DOOM [37]	GNN-based LDM	reward-based MPC

TSSM implies operating in the data-driven model-based domain, hence the focus of this chapter and Table 2.4 shows an overview of related literature and planning methods.

While all of these approaches use model-predictive control (MPC) to plan actions, the key distinction lies in how each method learns and represents cloth dynamics. Some approaches learn pixel-based visual dynamics models that predict future RGBD observations [43], others operate in learnt latent spaces, either optimised directly via contrastive objectives [40] or structured as graph-based representations. The latter includes models that infer latent connectivity graphs from partial observations [41] or build keypoint-centric graphs from depth inputs [37]. These diverse model-based strategies highlight different ways to represent and predict complex fabric dynamics for effective robotic manipulation.

Among these, PlaNet-ClothPick [9] inspired this thesis. It extends PlaNet's recurrent state-space model (RSSM) to pick-and-place fabric flattening, combining domain-specific constraints with a large offline dataset to improve performance. The next section reviews its design choices and key limitations, which motivate the adoption of a Transformer state-space model (TSSM) in this work.

2.4.3. Limitations of PlaNet-ClothPick

PlaNet-ClothPick adapts PlaNet's RNN-based world model to the fabric flattening setting, but faces additional challenges compared to rigid object manipulation. The complex, discontinuous shapes of cloth, particularly along folded or wrinkled contours, are difficult for an RSSM to model accurately, leading to blurred latent reconstructions that can misguide pick-point selection and cause failed manipulations. To mitigate these issues, the method introduces several targeted modifications. ClothMaskPick-MPC limits pick sampling to pixels identified as cloth via depth data or predicted masks, reducing the chance of mispicks. The reward function builds on that of Hoque *et al.* [43] to promote coverage improvements, add bonuses for high coverage, and penalise unhelpful behaviours such as excessive movements or unflattening. KL balancing [6] is applied to encourage priors to align more closely with posteriors, improving model stability. Finally, the system is trained offline on a diverse dataset of 1.12 million transitions, augmented to increase variability. These design choices together produce a compact, computationally efficient LDM that performs on par with mesh-based simulation methods while retaining faster inference.

Despite advances, two key limitations remain. PlaNet-ClothPick requires 10x more training data compared to SotA systems [9], indicating low data efficiency, and highlighting the difficulties in learning complex cloth dynamics using a RSSM. Second, the reconstructed observations from the predicted priors often have blurry cloth boundaries (see Figure 6.1), which impacts the ability for multi-step planning. Since pick locations are sampled from predicted priors during planning, sharp contours would help action selection.

2.4.4. Motivation for Transformer Replacement

Transformers offer an alternative architecture that may overcome core limitations of PlaNet-ClothPick in modelling cloth dynamics. First, they have shown data efficiency despite their model size [12], [14], [15], making them well-suited for leveraging large offline datasets. Second, their ability to model long-range dependencies has shown strong results in temporal sequence modelling and model-based reinforcement learning (MBRL) [11], [13]. The attention mechanism underlying the Transformer architecture has recently shown promising results in cloth-like deformable object (CDO) manipulation, used in *attention-based graph convolution* [37], which motivates using a full Transformer model further. Lastly, Mo *et al.* [39] applied Transformers to model-free RL, but prior work [41], [43] shows that model-based RL achieves better sample efficiency in CDO, supporting my focus on Transformer-based MBRL.

The application of a full Transformer-based LDMs in the specific context of CDO manipulation is novel and underexplored. The use of these recent advances could address challenges in accurate dynamics modelling and state prediction for CDOs, which motivates this thesis.

3. Methodology

This chapter focuses on the architecture and theoretical aspects of the recurrent state-space model (RSSM) and the Transformer state-space model (TSSM). Understanding both architectures is essential to make design choices in the TSSM for cloth-like deformable object (CDO) manipulation and use in the simulation framework AGENT-ARENA [16], aiming to align the Transformer-based model with the RSSM to ensure comparability. Beyond its methodological role, it provides a *theoretical framework*: a detailed theoretical exposition and comparative analysis of RSSM and TSSM, clarifying their mathematical foundations and representational differences in the context of cloth manipulation, and supported by novel figures illustrating their representational differences and the potential advantages of the TSSM in modelling complex cloth dynamics. While the RSSM is retained unchanged as the baseline model, I make design choices for each module of the TSSM, some matching those in the original TRANSDREAMER implementation, others adapted for the purposes of this study.

Extending on Section 2.2.4, Hafner *et al.* [5] and Kadi *et al.* [9] use the following terms, which this thesis will adopt and will be explained. The *Belief* h_t is the deterministic latent state evolved with each action, and represents the internal state representation maintained by the latent dynamics model (LDM), encoding temporal information from past states and actions. The *prior* $p(\tilde{z}_t \mid h_t)$ is a generative distribution that models the uncertainty of the latent state at time t, conditioned only on belief. The *Posterior* $q(\hat{z}_t \mid h_t, o_t)$ includes the uncertainty of the latent state at time t, if a true observation is available. It is a representation of the environment learnt from observations o_t at a specific time step. The *Latent state* $s_t = (h_t, z_t)$ is used to describe the lower dimensional representation consisting of the deterministic (h) and stochastic (z) component without specifying if it is a prior or posterior.

3.1. Recurrent State-Space Architecture

This section outlines the RSSM architecture to provide a basis for identifying differences with the TSSM and to inform my design choices for training a TSSM for cloth manipulation. The RSSM consists of four separate models that each serve different purposes. The transition model is responsible for modelling state transitions, encoder and decoder are responsible for compressing and reconstructing environment observations and the reward model learns to predict rewards for each state. This architecture, implemented in PlaNet-ClothPick, models dynamics via both *deterministic* and *stochastic* latent variables. These represent the state of the network at each time step t. The novelty of RSSM, including uncertainty modelling in the sequence modelling learning process was proposed by Hafner *et al.* [5] and used in PLANET-CLOTHPICK, while also providing the methodological foundation for the TSSM.

Figure 3.1 describes the overall model architecture, and the sequential generation of new latent states with every input observation and action pair, as well as the use of a previous latent state instead of an observation, which is used to predict into the future. The transition model is the central component that captures the complex evolution of the dynamics. The observation and representation models are represented by the green encoder and decoder modules, and the reward model acts independently to predict a scalar reward from

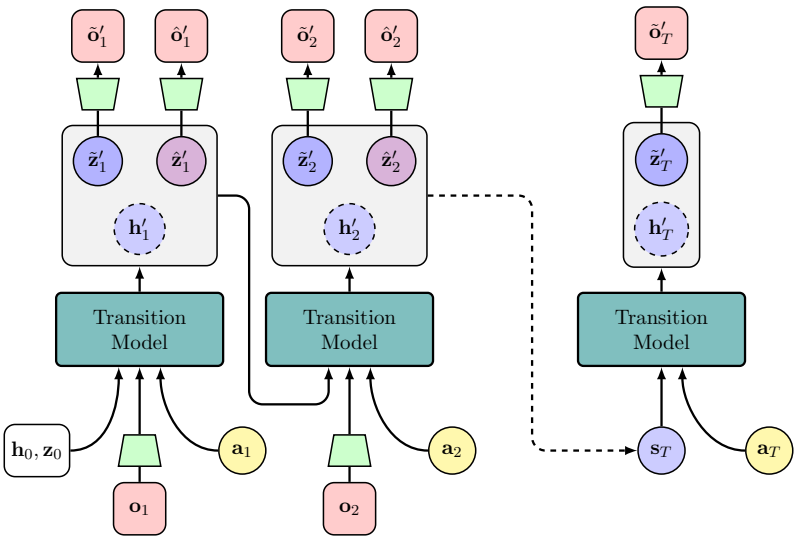


Figure 3.1.: Combined depiction of the sequential RSSM imagination rollout and internal transition model block structure. If observations are available, the posterior $\hat{\mathbf{z}}_t$ is passed forward into the next transition block, which is computed by embedding the belief and the observation. For inference (no observation available), the transition model produces $\tilde{\mathbf{z}}_t$ using only the deterministic belief instead of the posterior. The input $\mathbf{s}_T$ denotes either prior or posterior of the previous computation and the deterministic belief $\mathbf{h}_t$.

the current prior or posterior state $s_t = (h_t, z_t)$. Table 3.1 shows the number of parameters present in each model, which represents the relative sizes of the individual parts.

3.1.1. Transition Model

The transition model updates the hidden deterministic state h_t based on the previous deterministic state h_{t-1}, the previous stochastic state z_{t-1}, and the action taken at the previous step a_{t-1}.

$$h_t = f_{GRU}(h_{t-1}, z_{t-1}, a_{t-1}) \tag{3.1}$$
$$p(\tilde{z}_t|h_t) = FC(h_t) \tag{3.2}$$
$$p(\hat{z}_t|h_t, o_t) = FC(h_t, o_t) \tag{3.3}$$

where f denotes the function computed by the RNN and FC represents a fully connected neural network.

The RSSM transition model illustrated in Figure 3.2 uses a gated recurrent unit (GRU) [44] (specific type of recurrent neural network (RNN)) to capture the deterministic *belief state*, which include the long-term dependencies. A RNN is a neural network structure that feeds the output back into the latent layer and therefore recursively uses the whole sequence history [45], which is illustrated by the h_t nodes that originates directly from the GRU output of the previous step. GRUs are built on the gating mechanism introduced in the Long Short-Term Memory (LSTM) by Hochreiter *et al.* [46], which allows the network to decide how much input to consider and how much of the previous sequence to forget. GRUs benefit from a simplified architecture and are computationally more efficient, while still better at capturing long-term dependencies than previous RNNs [29]. The GRU cell used has 200 units. All other functions are fully connected layers of size 200.

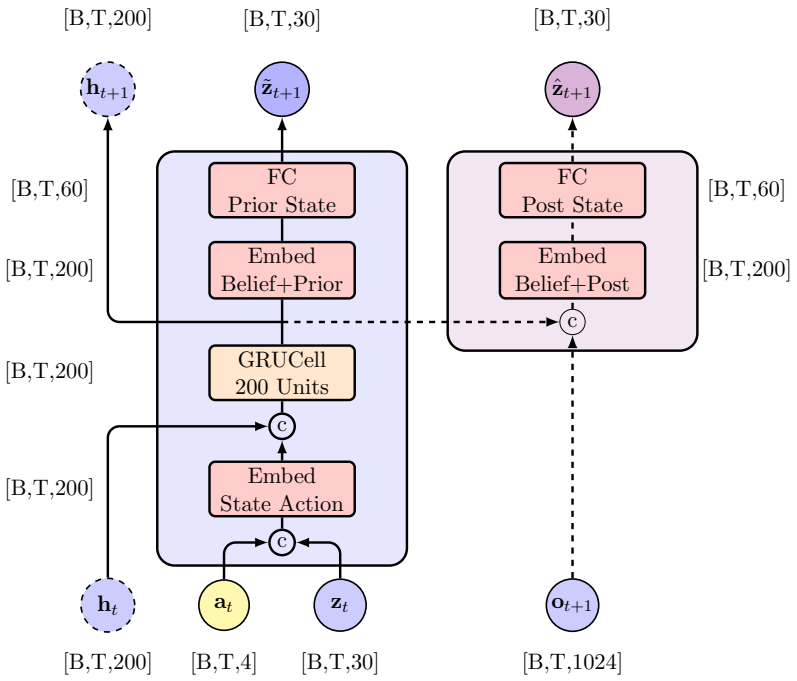


Figure 3.2.: The transition model of the RSSM. The node $\mathbf{h}_t$ represents the evolving deterministic belief, which is evolving with every observation and captures the temporal dependencies. $\mathbf{z}_t$ denotes the previous stochastic state, either a prior or posterior, and $\mathbf{a}_t$ the action. The dotted lines indicate the optional process of producing a posterior state, if embedded observations $\mathbf{o}_{t+1}$ are available. The output dimensions of each layer are listed on the side.

The stochastic part of the transition model defines the probabilistic latent state z_t by modelling a distribution conditioned on the deterministic state h_t and, where applicable, the observation. It consists of two components: the *prediction model* (Equation 3.2) and the *representation model* (Equation 3.3). The prediction model produces a prior distribution $p(z_t \mid h_t)$ based only on the current deterministic state h_t, modelling the expected latent dynamics without using the observation. It is implemented using a fully connected layer (labelled `FC Prior State` in Figure 3.2) that output the mean $\mu_{\text{prior},t}$ and standard deviation $\sigma_{\text{prior},t}$ of a Gaussian distribution. The representation model produces a posterior distribution $q(z_t \mid h_t, o_t)$ by incorporating the encoded observation. It uses a similar architecture (shown as `FC Post State`) and outputs the parameters $\mu_{\text{post},t}$ and $\sigma_{\text{post},t}$. Together, they form the stochastic transition component of the RSSM.

3.1.2. Encoder-Decoder Architecture

The encoder and decoder networks form the main interface between raw visual observations and the latent state of the RSSM. The encoder transforms each image into the compact embedding $o_{t+1} \in \mathbb{R}^{1024}$ in Figure 3.2, but this embedding alone is not the full latent state. Instead, it serves as an observation representation that is integrated with the predicted deterministic state h_{t+1} to produce a posterior latent state. This posterior state encodes not only visual features from the current observation but also accumulated information about the environment's physical interactions and long-term dynamics learnt over many trajectories. The decoder performs the reverse operation, reconstructing observations from the latent state to provide a reconstruction signal for training.

The encoder is part of the representation model $q(z_t \mid o_t)$, which maps input observations $o_t \in \mathbb{R}^{64\times64\times3}$ into a latent embedding space. The encoder consists of a stack of four con-

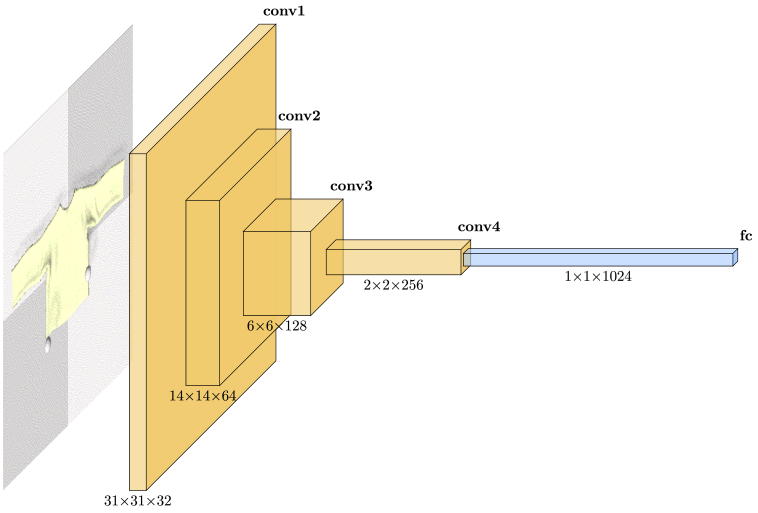


Figure 3.3.: The encoder transforms a raw image into a compact embedding, enabling efficient inference in the latent space. It compresses the image $\in \mathbb{R}^{64\times64\times3}$, which needs 12288 values for representation into a vector with 1024 values.

volutional layers with increasing channel dimensions and decreasing spatial resolution:

$$\text{Channels: } 3 \to 32 \to 64 \to 128 \to 256, \quad \text{Output: } 2 \times 2 \times 256$$

This feature map is then projected into a fixed-dimensional latent vector of size $1 \times 1 \times 1024$, using a fully connected layer (fc). This embedding is used as the observation representation in the posterior inference model. Batch normalisations and rectified linear unit (ReLU) activations follow each convolution. ReLU sets all negative outputs to zero, introducing non-linearity, which allows to learn more complex patterns while remaining computationally efficient.

The decoder is part of the observation model $p(o_t \mid h_t, z_t)$ and reconstructs image observations from the latent state $s_t = (h_t, z_t)$. It receives the concatenated latent vector and projects it into a structured feature map using a fully connected layer, reshaping it to a low-resolution feature volume.

A sequence of four transposed convolutional layers then upsamples the feature map to the original image resolution:

$$\text{Channels: } 128 \to 64 \to 32 \to 3 \text{ (RGB)}$$

These deconvolutions reconstruct the observation through these upsampling steps in a deterministic way, also including batch normalisations between layers.

The encoder and decoder together implement a bottleneck that forces the model to learn compact yet informative representations. The encoder compresses visual observations into a latent representation for the transition model, while the decoder ensures that the latent state remains accurate by using the reconstructed state as a training signal. This architecture is found in Hafner *et al.* [5], used in PlaNet-ClothPick [9], and will be used in the TSSM.

3.1.3. Reward Model

The reward model predicts the reward r_t given the current deterministic belief state h_t and the stochastic latent state z_t. Formally, the model predicts the conditional reward:

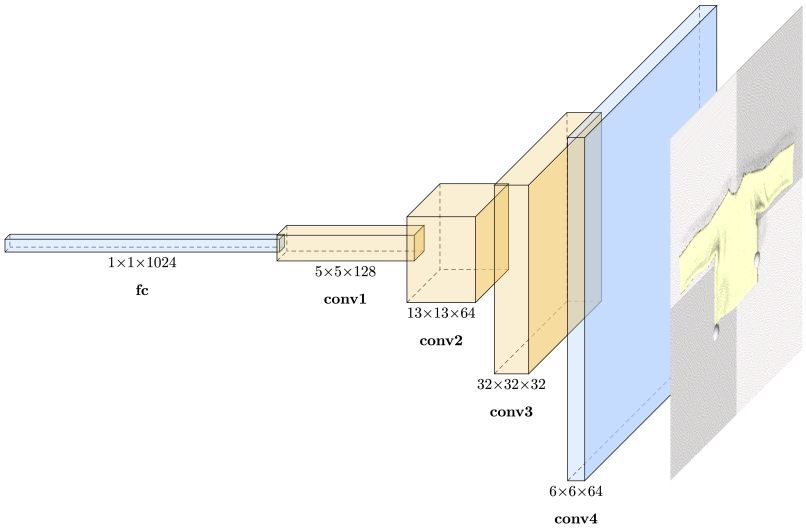


Figure 3.4.: The decoder reconstructs image observations from the latent state. It inverts the encoder process via transposed convolutions.

$$r_t = f_{MLP}(h_t, z_t) \tag{3.4}$$

This is implemented as a multi-layer perceptron (MLP) that produces a scalar value that represents the predicted reward. The model operates deterministically and does not model uncertainty in the reward signal.

The architecture consists of three fully connected layers. The input layer receives the concatenated belief and stochastic state vectors, resulting in an input of dimension 230. This is followed by two hidden layers, each with 200 units and ReLU activations. The final layer outputs a single scalar value corresponding to the predicted reward.

3.2. Transformer State-Space Architecture

This section presents the Transformer state-space model (TSSM), which – like the recurrent state-space model (RSSM) – comprises a transition model, encoder, decoder, and reward model. The transition model is described in detail, emphasising its key difference from Figure 3.2: TSSMs avoid sequential computation when evolving the deterministic state. This architectural change is central to the comparative analysis in Chapter 3 and is illustrated with novel figures to clarify representational differences. Building on this, the chapter outlines the methodological design choices I made to adapt the TSSM for the complex CDO domain.

The TSSM extends the RSSM by replacing the deterministic transition based on GRU with a Transformer. Like RSSM, it maintains both deterministic and stochastic latent components, but leverages attention to better capture long-range temporal dependencies. The overall architecture is illustrated in Figure 3.5, and the breakdowns of the parameters in comparison to the RSSM are shown in Table 3.1. In contrast to RSSM, the posterior $q(\tilde{z}_t|o_t)$ is modelled using only the state observation, and does not rely on the sequential update of the deterministic state.

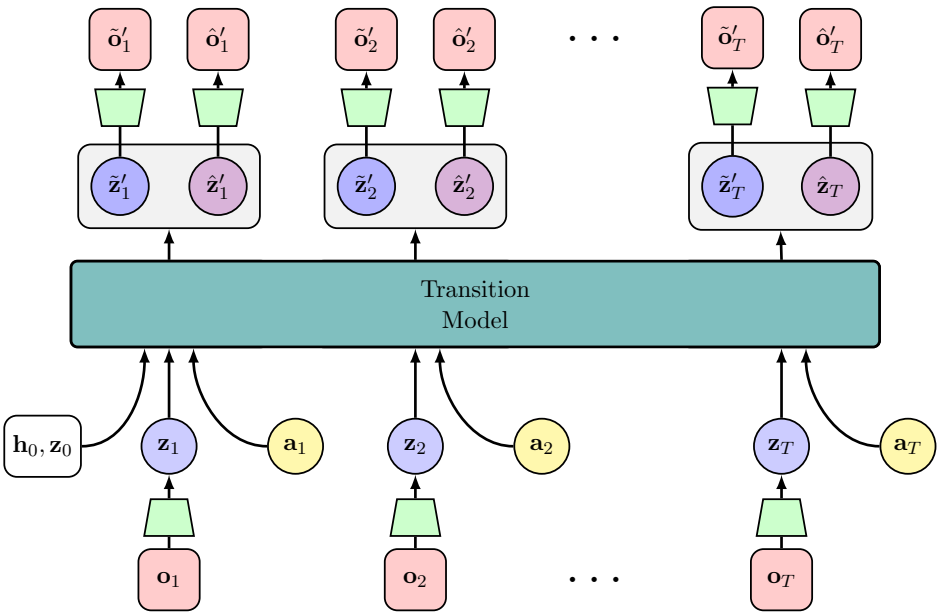


Figure 3.5.: The Transformer attends to all previous time steps to compute each output. At time t, the prior $\tilde{z}_t$ is computed using the full sequence of prior stochastic states $[z_1 : z_{t-1}]$ and actions $[a_1 : a_{t-1}]$. Observations are embedded and used only for computing the posterior. The deterministic belief h_t is internal to the model and not shown here. Inference (imagining future states) works iteratively by predicting a prior, adding it to the trajectory and feeding back into the transition model.

Table 3.1.: Number of parameters for each module in the PlaNet-ClothPick and TransDreamer agents. The transition model consist of additional neural components (MLPs, Embeddings) to the central RNN/Transformer that prepare the data before and predict the uncertainty after the transition computation.

Component	**PlaNet-ClothPick**	**TransDreamer**
Image Encoder	691,680	1,551,312
Transition Model	557,520	11,228,397
RNN	*241,200*	–
Transformer	–	*9,846,496*
MLPs, Embeddings	*316,320*	*1,381,928*
Image Decoder	3,799,014	8,977,299
Reward Model	86,601	734,801
Total Parameters	**5,134,815**	**22,491,836**

3.2.1. Transition Model

The transition model replaces the GRU with a Transformer that processes the full sequence of previous stochastic states and actions. Equation 3.7 indicates that the representation model is only dependent on the current observation, which allows the modelling of each stochastic state representation without having the evolve a deterministic component over all previous states. Unlike RSSMs local sequential updates, the Transformer accesses the full historical state and action context, enabling better modelling of long-range dependencies. This is illustrated by the sub-indices $s_{0:t}$, which represent a whole trajectory of states and actions that can be sent through the transition model and the new states $s_{1:t}$

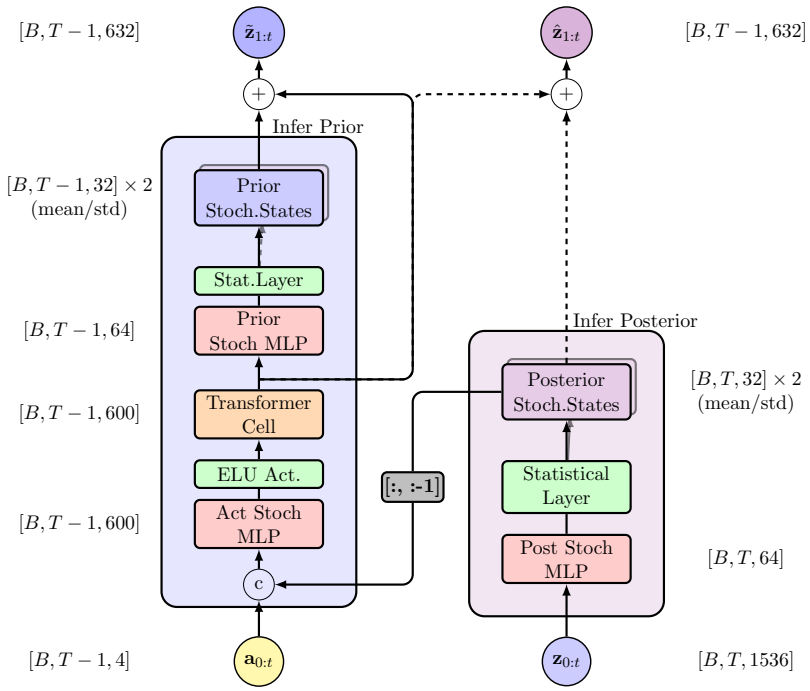


Figure 3.6.: TSSM Transition Model. The '+' indicates that the deterministic component (the Transformer output) is shared to produce the posterior / prior state by adding the deterministic component (Transformer output) to the stochastic states. The 'c' operation concatenates the actions with posterior states, temporally aligned to indices [0:T-1] to match the action length. The Transformer predicts the next state posterior for each state in the sequence, and the deterministic output sequence is in [1:T] after the cell. During inference, the input sequence $\mathbf{z}_{0:t}$ includes the predicted priors instead of posteriors.

are computed in parallel. The architecture generates multilayer representations that can be concatenated or selectively used for different tasks. The stochastic latent state z_t is modelled by the *prediction model* and *representation model*, shown on the left and right of Figure 3.6. The prior is inferred from the Transformer output via the `Prior Stoch MLP` and `Stat.Layer`, forming the prediction model (Equation 3.6) producing mean and standard deviation parameters of a Gaussian distribution. The posterior uses the current observation embedding and follows a similar structure, forming the representation model (Equation 3.7).

$$h_t = f_{\text{Transformer}}(z_{1:t-1}, a_{1:t-1}) \tag{3.5}$$
$$p(\tilde{z}_t \mid h_t) = \text{MLP}(h_t) \tag{3.6}$$
$$q(\hat{z}_t \mid o_t) = \text{MLP}(o_t) \tag{3.7}$$

Transformer Architecture The *transformer cell* in the TSSM, shown in Figure 3.6 in orange, is implemented as a decoder-only architecture to predict future states one step at a time based on previous outputs, a process known as autoregressive generation, while leveraging the strengths of Transformers over traditional RNNs. This means that unlike translation tasks that require encoder-decoder architectures to map between two different sequences, latent dynamics models only process a single sequence of past observations and imagine one future state, making the decoder-only option the most efficient. Table 3.2 shows the large size of the Transformer model, but this chosen architecture achieves the same training time as the smaller RSSM due to parallelisation advantages.

Table 3.2.: Parameter counts for the Transformer module that replaces the GRU module.

Variable/Representation	Parameters
Per MHA Layer	**1,230,000**
Query Projection	307,200
Key Projection	307,200
Value Projection	307,200
Output Projection	307,200
Layer Normalisation	1,200
Per Feed-Forward Layer	**1,231,624**
First Linear [600 $\rightarrow$ 1024]	615,424
Second Linear [1024 $\rightarrow$ 600]	615,000
Layer Normalisation	1,200
Per Transformer Layer	**2,461,624**
Total Transformer	**9,846,496**

Figure 3.7 shows the high-level layout of the chosen Transformer architecture for the TSSM. The input tensor, shaped $(T, B, 600)$, is first positionally encoded to provide both temporal and spatial context to the model [10]. This encoded input is then processed through a stack of four identical *Layers*. Each Layer consists of a LayerNorm (LN), multi-head attention (MHA), another LN, and a feedforward network (FFN), with residual connections after both the MHA and FFN sublayers. The architecture ensures the autoregressive behaviour via the use of an *Attention Mask*, which restricts the access of the input sequence such that each spatial location at time t can only attend to locations at times $\leq t$, ensuring that predictions depend only on past observations [47]. The output of the final Attention Block retains the shape $(B, T, 600)$, and the outputs from each of the four layers are concatenated along a new dimension, resulting in a final output of shape $(B, T, 4, 600)$, so that each layer output is accessible for later processing. I follow the approach used in recent Transformer-based LDMs that have outperformed RSSMs in modelling Atari games, where only the final Transformer layer output is used as the deterministic state for subsequent computation [12]–[14]. This differs from the original implementation of TransDreamer and reflects a design choice informed by the more recent successes of TSSMs.

In the multi-head attention (MHA) module, the input sequence is first projected into three separate representations: the query (Q), key (K), and value (V) matrices. These are obtained via independent learnt linear transformations, each capturing different aspects of the input features. The queries determine what each position seeks, the keys represent what each position offers, and the values carry the actual information to be aggregated. For each attention head, Q, K, and V have dimensions (T, B, d_q),(T, B, d_k) or (T, B, d_v), where T is the sequence length, B is the batch size, and d_q, d_k, d_v are the head-specific projection sizes, which are equal in this self-attention architecture. The MHA module reshapes the input for 8 attention heads, each with 64 dimensions. It computes independent attention score matrices for each head in parallel using Equation 3.8 with causal masking to prevent attending to future positions. This computational flow is shown in Figure 3.8a. The output is then concatenated and projected back into $(B, T, 600)$, which flows into the feed-forward layer.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{3.8}$$

The attention mechanism shown in Equation 3.8 and Figure 3.8b, produces a weighted sum of the value vector, where the weights are created by how good the query matches with the key vector. In this architecture, we use separate linear projection, each with its own weights, so the attention input is different for each input vector, with each linear

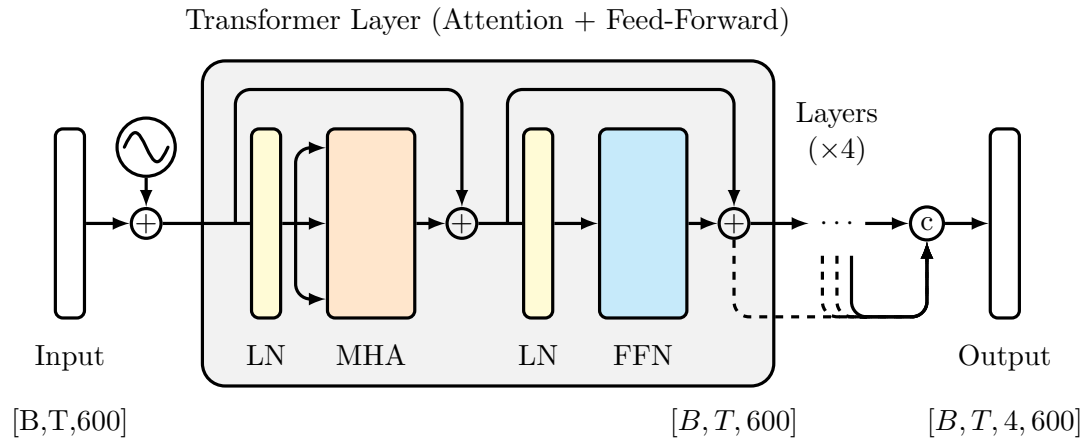


Figure 3.7.: Layout of the Transformer cell used in the transition model of the TransDreamer latent dynamics model (LDM). The input vectors, representing the deterministic latent state ($d = 600$), are first positionally encoded (sine symbol). The Transformer that will be used in TSSM consists of four identical layers stacked sequentially, each comprising a multi-head attention (MHA) block followed by a FFN, with pre-layer normalisation (LN) and residual connections. All intermediate layer outputs are concatenated to form the final multi-layer output. Following recent state of the art (SotA) Transformer-based LDMs, only the output of the last layer (solid arrow) is used as the deterministic state for subsequent computation. The Transformer illustration was inspired by Vaswani *et al.* [10].

layer learning different features of the latent state. Essentially, the Transformer output is similar to a weighted sum of the input sequence for each step.

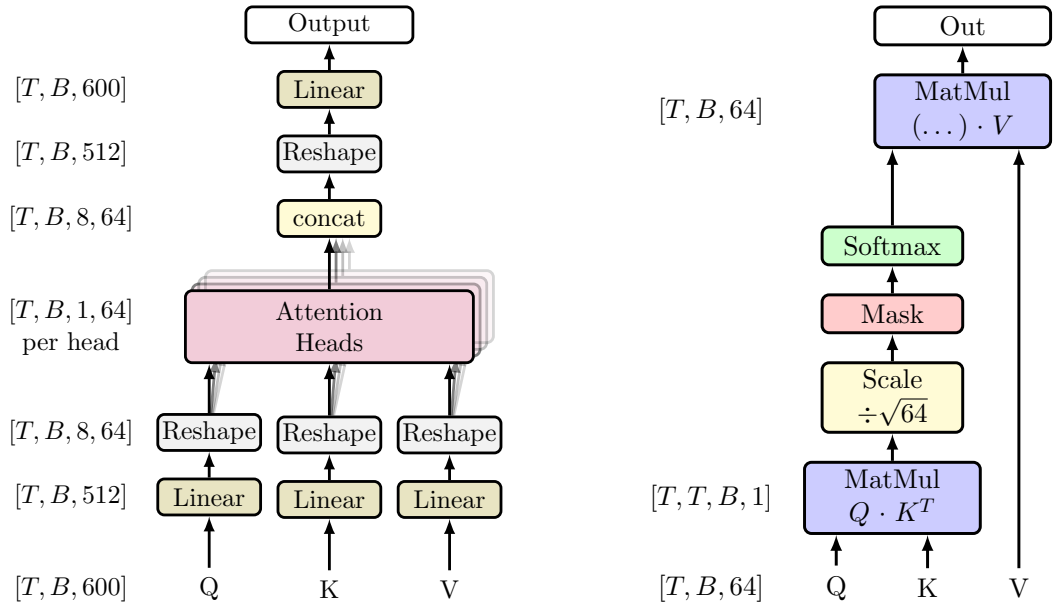
The feed-forward layer is implemented using a three layer MLP with the parameters specified in Table 3.2, using a ReLU activation function, indicated by the $\max(0, \cdot)$ notation in Equation 3.9. This module also applies pre-layer normalisation to the input, as illustrated in Figure 3.7.

$$\mathrm{FFN}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2 \tag{3.9}$$

For regularisation, *dropout* [48] is applied to each of the layers after the feed-forward block. Dropout randomly deactivates a subset of neurons during training, encouraging robust and generalised learning outcomes and avoids reliance on specific neurons.

To summarise, the new transition model architecture now features an autoregressive Transformer-decoder block, that can attend to all past states in order to predict future deterministic latent states. The stochastic component is computed similarly to the RSSM by using an MLP to predict the stochastic distribution over the deterministic state, as shown in Equation 3.6 and Equation 3.7. Furthermore, the Transformer has access to the entire history (up to 2000 steps) passed to it. Each state at position t represents the deterministic component given the complete history up to time step t, which is denoted in Equation 3.5. This is fundamentally different from RNNs, which process data sequentially. The Transformer computes all T deterministic representations in one forward pass, with each one having appropriate historical context through the causal attention mechanism.

Given this new architecture that allows the TSSM to compute entire sequences in parallel, the following section describes the design choices I could make to adapt the TSSM from TransDreamer to the cloth domain. The computational efficiency allows to train a larger model in the same time on the same hardware as PlaNet-ClothPick, allowing



(a) Multi-Head Attention.

(b) Single Attention Head.

Figure 3.8.: In multi-head attention as illustrated in (a), queries, keys, and values are linearly projected and fed into parallel attention heads. The outputs from these heads are then concatenated and linearly projected to form the final result, enabling the model to process different features in each attention head. The attention head in (b) maps a query and a set of key-value pairs to an output by computing the dot products of the query with all keys, and applying Softmax to compute the *weight matrix*, represented by the output dimension [T,T,(...)]. The following mask ensures that only past states have non-zero entries for each value. The final output is then a weighted sum of the weight matrix and the values (V). The Batch and Time dimensions are swapped for this computation only. The graphic is inspired by Vaswani *et al.* [10].

to carry over some of the original design choices into the cloth domain, while keeping it computationally competitive.

3.2.2. Design Choices for CDO Adaptation

I adapted TSSM to the cloth domain with modifications aimed at increasing representational capacity while maintaining comparable training times. A central change was to exploit the ability of the TSSM to compute posteriors from visual input alone, enabling full-sequence parallel training and enhancing its efficiency when trained on substantial datasets. This removed sequential update dependencies present in the RSSM and allowed an increase of the deterministic latent dimension from 200 to 600. Despite this increase and the significantly larger model size (Table 3.1) the training time was comparable to PlaNet-ClothPick, justifying this change. Recent Transformer-based models that have been successful in Atari gameplay modelling have used a similar hidden size [12], [13].

The stochastic state dimension was kept at 32, similar to the size 30 of RSSM, preserving comparability, and one distribution type was added: a Gaussian distribution, as in PlaNet-ClothPick, which is well-suited to the continuous action space of the CDO. The original categorical distribution from TransDreamer is retained for ablation studies.

The Transformer depth was reduced from six to four layers, following evidence from Zhang *et al.* [12] indicating no performance improvement beyond four layers in Atari modelling tasks with similar image input sequences. I will be conducting additional experiments with two layers to explore potential efficiency gains. Other hyper-parameters, including layer width and attention head configuration, were kept identical to TransDreamer to preserve tested stability in training.

The encoder and decoder architectures were retained in their original TransDreamer form. The encoder processes $64 \times 64 \times 3$ RGB observations using four convolutional layers with channel progression

$$3 \to 48 \to 96 \to 192 \to 384$$

each followed by ELU activations without batch normalisation. ELUs apply a linear activation for positive inputs and a smooth exponential curve for negative inputs, mitigating vanishing gradients while producing mean activations closer to zero. I decided to keep the ELU activation, as it has been shown to achieve faster learning and better generalisation performance than ReLUs [49]. The final $2 \times 2 \times 384$ feature map is flattened and linearly projected into a 1536-dimensional embedding, which serves as input to the posterior inference model. The decoder mirrors this process in reverse: a dense projection from the latent state $s_t = (h_t, z_t)$ into a $2 \times 2 \times 384$ feature map, followed by four transposed convolutional layers with channel progression

$$384 \to 192 \to 96 \to 48 \to 3$$

also with ELU activations and no batch normalisation. The decoder outputs a diagonal Gaussian distribution over pixel intensities with fixed $\sigma = 0.5$, where σ controls the assumed uncertainty (spread) of each predicted pixel value. A fixed value avoids the need for the network to predict variance explicitly, simplifying training and preventing instability from small or large variance estimates. Retaining this architecture ensured compatibility with the CDO dataset resolution and leveraged a design already validated for stable training in TransDreamer.

I have chosen three variants for reward modelling: (i) the default TransDreamer stochastic model (734,801 parameters) outputting a Gaussian over scalar rewards, (ii) a deterministic scalar MLP, and (iii) the smaller RSSM reward model from PlaNet-ClothPick (86,601

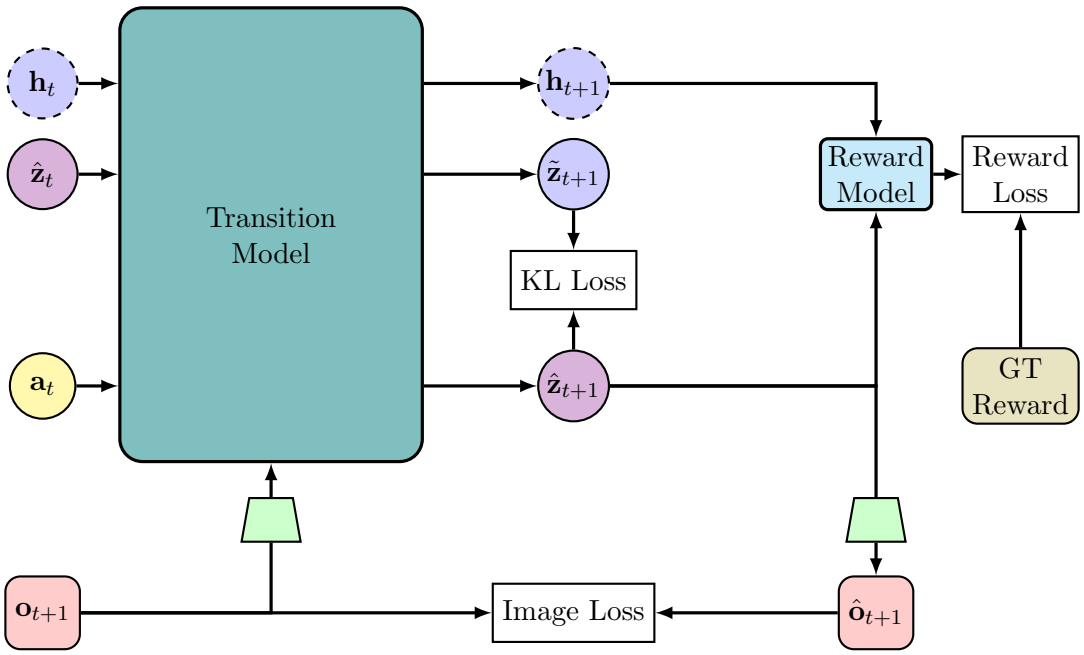


Figure 3.9.: Training-time computational flow of both LDMs, comprising an encoder for observation embedding and decoder (green trapezoids), the transition model, and the reward model. The loss terms used to train the model include the image reconstruction loss, reward prediction loss, KL divergence loss between prior and posterior.

parameters). The default stochastic model receives the concatenated belief and stochastic state vectors, processes them through two fully connected layers of 600 units with ELU activations, and outputs the parameters of a Gaussian distribution over scalar rewards. It is expected to better capture the temporally extended and non-linear reward dependencies characteristic of cloth manipulation, while the others will be used for ablation studies.

Collectively, these adaptations constitute a novel architectural design of a Transformer-based LDM, developed to ensure stable and efficient training, enable targeted ablation studies, and maintain direct comparability with the RSSM baseline when modelling the complex spatial–temporal dependencies of cloth dynamics.

3.3. Latent Dynamics Training Objective

Following the introduction of RSSM and TSSM, the loss objectives are defined to ensure that LDMs learn accurate environment dynamics while allowing theoretical comparison between architectures. While the general structure of these objectives is similar, their exact implementation varies between TransDreamer and PlaNet and this section outlines the theoretical structure of the loss computation and my choices for TSSM at the end. Figure 3.9 shows how the different loss components are produced during model training.

The image reconstruction loss measures how well the model can reproduce the ground truth observation from its latent state. This is typically formulated as the negative log-likelihood of the observation given the latent representation. If the likelihood model is Gaussian, this formulation is equivalent to using a mean squared error objective, as in the original RSSM formulation [5]. TransDreamer, however, retains the log-likelihood formulation directly, making the approach agnostic to the specific choice of observation distribution and more easily adaptable to non-Gaussian cases.

$$-\mathbb{E}_{q(z_t|o_{1:t},a_{1:t})}\big[\log p(o_t|\hat{z}_t)\big] \tag{3.10}$$

The *reward loss* measures the accuracy of predicted rewards based on the latent state. Conceptually, this can be expressed either as a negative log-likelihood under an assumed reward distribution, or, if a Gaussian is assumed, as a mean squared error. The original RSSM formulation adopts the MSE view, which implicitly fixes the reward distribution to Gaussian. TransDreamer preserves the log-likelihood form, maintaining flexibility to model different reward distributions without altering the loss definition.

$$-\mathbb{E}_{q(z_t|o_{1:t},a_{1:t})}\big[\log p(r_t|\hat{z}_t)\big] \tag{3.11}$$

The Kullback and Leibler (KL) divergence loss aims to approximate the predicted prior distribution and the posterior distribution inferred from observations. It does this by measuring the KL divergence between the posterior at time t and the one-step prior predicted from the state at $t-1$ and action at $t-1$. The expectation is taken over z_{t-1} because the prior $p(\tilde{z}_t \mid z_{t-1}, a_{t-1})$ is conditioned on the previous latent state, requiring a sample from $q(\hat{z}_{t-1} \mid o_{1:t-1}, a_{1:t-1})$ to evaluate the KL divergence at time t

$$\mathbb{E}_{q(z_{t-1}|o_{1:t-1},a_{1:t-1})}\Big[\mathrm{KL}\left[q(\hat{z}_t|o_t) \,\|\, p(\tilde{z}_t|z_{t-1},a_{t-1})\right]\Big] \tag{3.12}$$

If the dynamics model is not very accurate, forcing the learnt representation (posterior state) to be close to its prediction (prior state) might make the learnt representation less accurate, therefore KL balancing [7] aims to reduce the amount the posterior is regularised to be similar to a predicted prior. It prioritises learning an accurate dynamics model by applying larger gradient steps to the part of the KL loss that tries to make the prior state prediction match the learnt posterior state. This improves the model's ability to make reliable multistep predictions, which is crucial for effective planning [7].

In summary, Equation 3.13 contains the full training objective, which is a sum of the individual loss components.

$$\begin{aligned}\mathcal{L}_{\text{PlaNet}} = \sum_{t=1}^{T} \Bigg(& -\mathbb{E}_{q(z_t|o_{1:t},a_{1:t})}\Big[\log p(o_t \mid \hat{z}_t) + \log p(r_t \mid \hat{z}_t)\Big] \\ & + \mathbb{E}_{q(z_{t-1}|o_{1:t-1},a_{1:t-1})}\Big[\mathrm{KL}\Big(q(\hat{z}_t \mid o_t) \,\|\, p(\tilde{z}_t \mid z_{t-1}, a_{t-1})\Big)\Big]\Bigg)\end{aligned} \tag{3.13}$$

In summary, my design decisions for the TSSM loss formulation prioritise flexibility, comparability, and suitability for ablation studies. The image reconstruction loss is expressed in log-likelihood form to remain agnostic to the assumed observation distribution, facilitating potential extensions to non-Gaussian cases. For the reward loss, both log-likelihood and mean squared error (MSE) variants are considered, enabling controlled comparison across Gaussian, non-Gaussian, and distribution-free reward model outputs as described in the previous section.

3.4. Project Management

To ensure efficient progress, the project was broken down into manageable tasks with clearly defined milestones, dependencies, time estimates, and risk assessments, following recommendations in Sommerville [50]. I adopted the *Kanban* methodology, which provided a clear visual overview of the project scope and the flexibility to re-plan when necessary [51]. The setup used Trello[1] for task tracking and TeamGantt[2] for visualising overall progress, with both tools integrated to remain synchronised automatically. Tasks were labelled according to primary and secondary objectives, and organised into *TODO* (due within a week), *IN PROGRESS* (ongoing larger tasks), and *DONE* (completed in the past week). I held weekly replanning sessions to review progress and schedule upcoming work, aided by Trello automations that moved imminent tasks into the *TODO* list. The preliminary work plan showing the individual tasks is provided in Appendix C. Evidence of the success of this methodology is found in the created project artefacts: The git history contains a comprehensive list of changes, demonstrating the iterative work done throughout this thesis, and weekly progress reports, as well as meeting notes, document the alignment of activities with the work-plan, ensuring that all objectives were met on time.

[1] https://trello.com/
[2] https://www.teamgantt.com/

4. Implementation

This chapter details the practical implementation of the design choices introduced in the methodology, focussing on my integration of the Transformer state-space model (TSSM) into PlaNet-ClothPick and the existing evaluation framework Agent-Arena [16]. The work goes beyond a direct reuse of existing models: I extracted the TSSM from TransDreamer and modified it to operate within the PlaNet-ClothPick codebase, as well as embedded into the Agent-Arena framework, creating a reusable implementation of the TSSM for cloth-like deformable object (CDO) manipulation. These choices, motivated by the unique challenges of modelling CDO dynamics, required architectural adaptation, forming a central novel contribution of this thesis.

The implementation addresses two main challenges. First, achieving stable training of the TSSM in the CDO domain by modifying both its architecture and training pipeline. Second, embedding the adapted model into Agent-Arena in a way that is fully compatible with its simulation, data handling, and control components. The remainder of this chapter details the software environment, the integration process, and model-specific implementation decisions that enabled these contributions.

4.1. Development Environment and Tools

I used Docker and Conda to create a reproducible experimental environment, adapted for Ubuntu LTS versions *22.04* and *24.04*. These environments support GPU acceleration through CUDA, compatible with a *NVIDIA RTX 3060 GPU*. Package dependencies for deep learning, simulation, and vision tasks were managed through these Conda configurations. To ensure robust remote training over SSH connections, *tmux* was used, allowing background process execution and session management. This setup facilitated reliable and flexible experimentation for robotics-based deep reinforcement learning tasks. The details of the environment setup process can be found in Appendix D.

4.1.1. Version Control and Reproducibility

I used *Git*[1] to track changes in the coding project. To ensure an organised approach, the *GitHub* Workflow[2] was adopted, using a *develop* branch for active work, merging each larger working state into the *main* branch, and *feature* branches to work on milestones, which are merged into *develop*. Every objective, once met, was marked using an annotated git tag to ensure it can be reproduced, given the environment is set up.

4.2. Agent-Arena Framework

Agent-Arena [16] is an existing open-source Python framework designed to evaluate control algorithms in robotics research and I use this for all experiments conducted with TSSM.

[1] https://git-scm.com/
[2] https://docs.github.com/en/get-started/using-github/github-flow

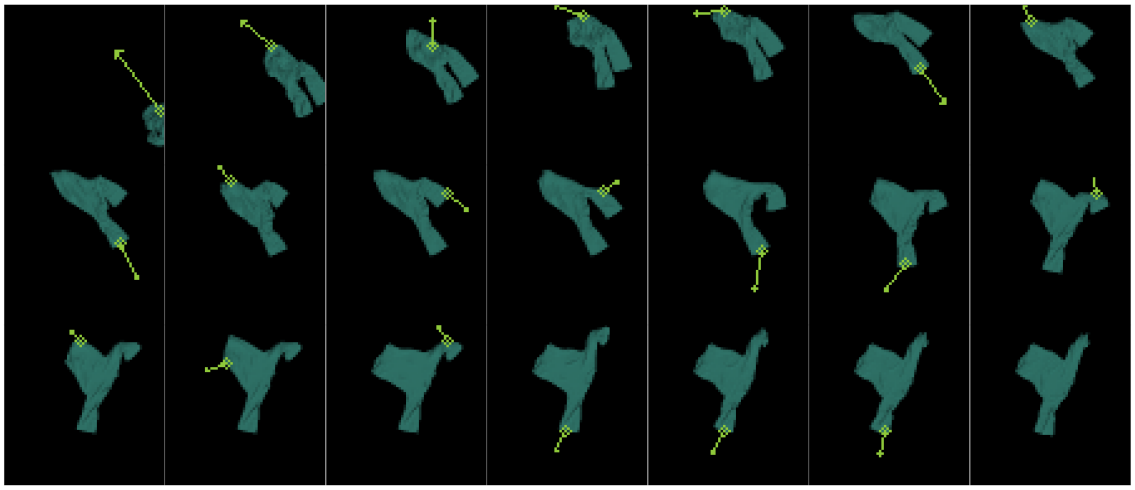


Figure 4.1.: This is an example of a full 21 step rollout of a trajectory from the garment dataset, including the actions taken (the arrow starts at the "pick" point and ends at the "place" location) at each step to produce the next observation. Such trajectories are used for training and evaluation.

AGENT-ARENA unifies agent–environment interaction (Fig.2.1) and supports benchmark environments such as *SoftGym* [36], *Raven* [52], *DeepMind Control Suite* [53], *MuJoCo* [54], *PyBullet* [55], and *NVIDIA Flex* [56]. Its decoupled design separates the `Agent` (decision-making policy) from the `Arena` (environment), enabling flexible integration of domain-specific or general, analytical or learning-based methods. Agents are configured via YAML files and interact with arenas through a standard interface (`reset`, `act`, `init`, `update`), with arena-aware methods allowing context-dependent behaviour. The `TrainableAgent` subclass extends this to learning-based methods such as model-based reinforcement learning (MBRL), adding checkpoint save/load functions and a `train` method for multi-arena training. Checkpointing is controlled by update steps, allowing intermediate model weights to be stored during training. In this thesis, all TSSM experiments are implemented within AGENT-ARENA.

While this section outlines the existing framework, my contribution includes extending its `TrainableAgent` interface to accommodate the TSSM training and planning requirements, ensuring integration with AGENT-ARENA's architecture. This thesis will use *Agent-Arena* to conduct training, evaluation, tests and benchmarks, and wrap the TSSM into a usable class, inheriting from `TrainableAgent` for conducting evaluation. The dataset used will be explained in the following section.

4.3. Dataset

This section describes the cloth folding dataset collected by Kadi *et al.* [9], which was provided to me for training and evaluating the TSSM. The dataset consists of goal-conditioned visuomotor manipulation trajectories for cloth flattening tasks, and will be used after preprocessing and augmentation to train the new TSSM model that was adapted for the CDO domain. An example trajectory is shown in Figure 4.1.

The existing cloth dataset contains 40,000 trajectories of cloth manipulation episodes, with each trajectory consisting of 21 time steps (20 actions + 1 terminal observation). The total dataset comprises 840,000 time steps of interaction data. The dataset features a multimodal, goal-conditioned observation space designed to support complex visuomotor learning tasks involving deformable object manipulation. Each observation includes

Table 4.1.: Dataset Statistics

Property	**Value**
Total Trajectories	40,000
Trajectory Length	21 time steps
Total time steps	840,000
Dataset Split Ratios	[1%, 1%, 98%] (eval/val/train)

Table 4.2.: Dataset Entries, Dimensions and Values Ranges.

Entry	**Dimensions**	**Range**	**Description**
`rgb`	$(B, T, H, W, 3)$	$[0, 255] \subset \mathbb{N}$	RGB images
`mask`	$(B, T, H, W, 1)$	$\{0, 1\}$	Segmentation masks
`goal-rgb`	$(B, T, H, W, 3)$	$[0, 255] \subset \mathbb{N}$	Goal RGB images
`goal-mask`	$(B, T, H, W, 1)$	$\{0, 1\}$	Goal segmentation masks
`action`	$(B, T-1, A)$	$[-1, 1] \subset \mathbb{R}$	Pick and place coordinates
`reward`	$(B, T, 1)$	$[0, 0.5] \subset \mathbb{R}$	Scalar reward values
`terminal`	$(B, T, 1)$	$\{0, 1\}$	Terminal flags (episode done)

several components: RGB images ($\mathbf{o}_t^{rgb} \in \mathbb{R}^{64\times64\times3}$) representing the current visual state with pixel values in the range $[0, 255]$; binary masks ($\mathbf{o}_t^{mask} \in \mathbb{R}^{64\times64\times1}$) highlighting cloth regions with values in $[0, 1]$; and corresponding goal specifications in the form of a target RGB image ($\mathbf{g}^{rgb}$) and a goal segmentation mask ($\mathbf{g}^{mask}$) of matching dimensions. Each observation includes a scalar reward signal ($r_t \in \mathbb{R}$) quantifying progress toward the folding objective, bounded in the range $[0, 0.5]$, and a binary terminal flag ($d_t \in 0, 1$) indicating episode completion. The continuous action space consists of normalised pick-and-place operations represented as a 4D vector $\mathbf{a}_t = [x_{\text{pick}}, y_{\text{pick}}, x_{\text{place}}, y_{\text{place}}] \in \mathbb{R}^4$, where each coordinate is in the range $[-1, 1]$, denoting the location of the pixels to grasp and place. Overall, this large-scale dataset offers extensive coverage for training and evaluating visuomotor policies, particularly for tasks requiring precise manipulation of deformable objects and Table 4.2 shows an overview.

4.3.1. Data Preprocessing

This section outlines my custom `preprocess()` method I implemented to adapt raw dataset episodes for TSSM training, resolving structural mismatches between the dataset format and the Transformer-based dynamics model. This required sequence alignment, trimming, and value preparation that differ from the recurrent state-space model (RSSM) pipeline, ensuring that the architecture receives temporally consistent input tensors.

To ensure temporal consistency for model training, observations are trimmed by removing the final time step, and rewards, terminal flags are trimmed by removing the first time step, reducing all sequences to uniform length $T - 1$ and allowing proper tensor concatenation during neural network computation. Table 4.3 shows the correct alignment. The TransDreamer agent also requires different value preprocessing compared to the RSSM agent. It expects the data ranges listed in Table 4.2, and does normalisation and conversion internally. The agent expects observations under the configurable `input_type` key, which corresponds to RGB observations from the `'rgb'` key in the dataset. Unlike RSSMs observation fusion strategies that create concatenated representations of rgb and mask input, TransDreamer processes RGB observations directly, requiring the mask

Table 4.3.: Temporal alignment of observations, actions, rewards, and done signals during training. Rewards and terminals (d) start at step 1, because reward is only awarded for actions producing a new state. Actions and observations end at $T-1$, that last state will be inferred from a_{t-1} and produces r_t, the done flag would be '1' here. The last observation [(X)] is technically available, but for training all sequences need to be the same length so it will be trimmed.

t (time step)	0	1	2	3	4	5
o (observation)	X	X	X	X	X	(X)
a (action)	X	X	X	X	X	-
r (reward)	-	X	X	X	X	X
d (terminals)	-	X	X	X	X	X

only for the pick constraints in the model-predictive control (MPC) algorithm. The image encoder applies the normalisation transformation obs/255 − 0.5 internally. Actions, rewards, and termination flags undergo tensor conversion to ensure proper data types and device placement, but no mathematical transformations are applied. The temporal structure is preserved in the natural $(B, T, \ldots)$ format the architecture expects, eliminating the need for axis swapping operations that were necessary for RSSMs. These changes were necessary because the PlaNet-ClothPick preprocessing assumed an RSSM input format; I re-designed the preprocessing pipeline so that the TSSM agent can be trained within the same codebase without altering core dataset storage or loading logic.

4.3.2. Data Augmentation

Data augmentation includes random rotations, flipping, and adding noise, which adds variation to the observation space. It is used in MBRL to improve generalisation and mitigate *domain shift*, which refers to the differences between the simulated and the real world Markov decision process (MDP) [3]. It also supports more efficient training and aligns with the goal of learning transferable dynamics models[25]. For RSSM training, augmentation was applied uniformly across each full 21-step trajectory, with actions and observations transformed together to preserve temporal coherence and avoid breaking causal structure. For consistency, I reuse the same augmentation operations for the TSSM, applying them at the trajectory level. This setup enables controlled experiments in Chapter 5 with and without augmentation for ablation purposes.

4.4. Transformer State Space Model Implementation

To enable the TSSM from TransDreamer to operate within the PlaNet-ClothPick codebase and be deployed through the Agent-Arena framework, I first had to resolve fundamental mismatches in data flow, model interfaces, and execution context. This required analysing both systems' assumptions about input/output structure, temporal alignment, and latent state handling, and then adapting the `TransformerWorldModel` and `TransformerDynamic` components of TransDreamer so they could operate under Agent-Arena's `TrainableAgent` interface without altering simulation or dataset logic. These adaptations went beyond code translation: they preserved CDO-specific temporal coherence, ensured the TSSM latent dynamics remained compatible with the Cloth-Mask MPC planner, and maintained compatibility with the PlaNet-ClothPick training pipeline. This constitutes one core contribution of this thesis: creating a modular, reusable implementation of a TSSM for cloth manipulation, that can be used within the

```
action = {
    'norm-pixel-pick-and-place': {
        'pick_0': [0.2, 0.3],    # values from indices [0,1]
        'place_0': [0.5, 0.7]    # values from indices [2,3]
    }
}
```

Figure 4.2.: Action Dictionary for Planning Algorithm

framework for evaluating control algorithms and enables future research on Transformer-based dynamics models in CDO manipulation.

Algorithm 1 illustrates the adapted offline training loop. The implementation reuses the dataset loaders PlaNet-ClothPick but incorporates my custom preprocessing and trajectory handling in `compute_loss()`, ensuring the TSSM receives inputs in the correct temporal format and scale while maintaining alignment with the requirements of CDO tasks. To enable planning and acting in simulation, I extended the integration layer

Algorithm 1 Offline Training Loop

Require: Datasets $\mathcal{D}_{\text{train}}$, number of update steps U
Ensure: Trained model parameters

Initialize iterator over training data
$u_{\text{end}} \leftarrow \min(\text{global\_step} + U, \text{total\_steps})$
for $u = \text{global\_step}$ **to** u_{end} **do**
    global_step $\leftarrow$ global_step + 1
    Load episode b from iterator
    $traj \leftarrow \text{preprocess}(b)$
    $\mathcal{L}_{\text{model}} \leftarrow \text{compute\_loss}(traj, T)$ ▷ compute model loss
    $\text{grad\_norm} \leftarrow \text{backpropagate}(\mathcal{L}_{\text{model}})$ ▷ update model weights
    if Using warm-up schedule **then**
        Update Transformer learning rate
    end if
    if global_step mod `log_every_step` $= 0$ **then**
        Log loss, learning rate, gradient norms
    end if
    if global_step mod `checkpoint_every_step` $= 0$ **or** global_step $= 1$ **then**
        Save model checkpoint
    end if
end for
return True

with `act`, `init()`, and `update()` methods so the TSSM could maintain its latent state between environment steps. This allows the model to interface directly with the ClothMask MPC planner without altering its algorithmic logic. As shown in Algorithm 2, `planning_algo.act()` generates candidate actions in the native 4D vector format of the dataset, while `replace_action()` converts them into the structured dictionary format required by the planner (Fig. 4.2). Bridging the latent rollouts of TransDreamer with the action-selection interface of Agent-Arena required additional helper functions. `unroll_action_from_cur_state` evaluates each candidate action by unrolling the next latent state from the current one, and the custom `cost_fn()` ranks candidates based on average predicted rewards. These functions operate without modifying the planner's interface, using the `info` dictionary to pass necessary state data such as masks and observations.

Algorithm 2 Offline Action Selection using TransDreamer

Require: info_list: list of states as `dict`, update: boolean flag
Ensure: actions: list of actions as `dict`

actions ← empty list
for each $info$ in $info\_list$ **do**
    $action$ ← planning_algo.act([$info$]) ▷ calls unroll function from model
    $plan\_state$ ← planning_algo.get_state()
    for each (k, v) in $plan\_state$ **do**
        internal_states[info[arena_id]][k] ← v
    end for
    ret_action ← copy of config.action_output converted to dict
    ReplaceAction($action$, ret_action) ▷ converts action list into dict
    append ret_action to $actions$
end for
return $actions$

In summary, this chapter described the integration of the Transformer state-space model from TransDreamer for cloth-like deformable object manipulation within the PlaNet-ClothPick and Agent-Arena frameworks. This work involved designing a TSSM-specific preprocessing pipeline, preserving temporal coherence through augmentation, and restructuring the model to conform to the `TrainableAgent` interface while maintaining compatibility with existing planners. In doing so, the implementation addressed both the redesign and integration of the TSSM within PlaNet-ClothPick and its embedding into Agent-Arena, aligning with the contributions in Chapter 1. The resulting system is a modular, reusable foundation for future Transformer-based dynamics research in CDO manipulation and underpins the experimental evaluation in the next chapter.

5. Experiments

This chapter evaluates the performance of the new Transformer state-space model (TSSM) on garment flattening tasks, comparing the implementation described in Chapter 4 to the recurrent state-space model (RSSM)-based PlaNet-ClothPick. These experiments demonstrate the first application of a TSSM agent to cloth-like deformable object (CDO) manipulation, assessing its ability to model deformable object dynamics through reconstruction accuracy, reward prediction, and planning performance. The evaluation focuses on reconstruction accuracy, reward prediction, and planning, providing quantitative evidence of the model's suitability for capturing the complex dynamics of CDO environments.

5.1. Experimental Setup

The experiments use the TSSM variants introduced in Section 4.4 and summarised in Table 5.2, along with the RSSM-based PlaNet-ClothPick baseline, to enable a direct comparison of the adapted architecture against prior work. All models in Table 5.2, together with the baseline PlaNet-ClothPick, were trained on the same garment-flattening dataset (Section 4.3) using 50,000 batches of 50 trajectories. Original trajectories of length 21 were concatenated into sequences of 50, with termination indices marking new episodes. This setup exposes the model to 2.5 million transitional steps. The training covered three full dataset passes, promoting exposure to diverse data and improving generalisation [25]. Hyperparameters are listed in Table 5.1. Models were trained offline, following prior work [9], [37], [41], [43], due to higher sample efficiency when a fixed dataset is available [4]. Training was conducted on a single Nvidia GTX 3060 (see Appendix D), averaging 26 hours per model.

The trained TSSM variants differ in three key design choices defined in Chapter 3: (i) stochastic state distribution type — categorical (as in TransDreamer) vs. normal (as

Table 5.1.: The model was trained using an adaptive learning rate to ensure more stable convergence, which was reduced over the first half of the training batches. Gradient clipping was applied because initially, huge gradients have led to unstable training. The optimiser choice adopted from TransDreamer.

Hyperparameter	**Value**	**Description**
Optimiser	AdamW	With eps=1e-5, weight_decay=1e-6
Base Learning Rate	2e-4	Initial learning rate
End Learning Rate	1e-4	Final learning rate after decay
Decay batches	25,000	Steps over which learning rate decays
Total Training batches	50,000	Total number of training iterations
Batch Size	50	Training batch size
Sequence Length	50	Length of sequences used for training
Gradient Clipping	1,000	Maximum gradient norm

Table 5.2.: Model configurations for selected TSSM trainings. The design choices discussed in Chapter 3 are listed to provide an overview of the trained models.

Parameter \ Model	**05-07**	**07-07**	**21-07**	**23-07**	**01-08**
stochastic distribution	normal	categorical	normal	categorical	normal
augmentation	no	no	yes	yes	yes
Transformer layers	4	4	4	4	2

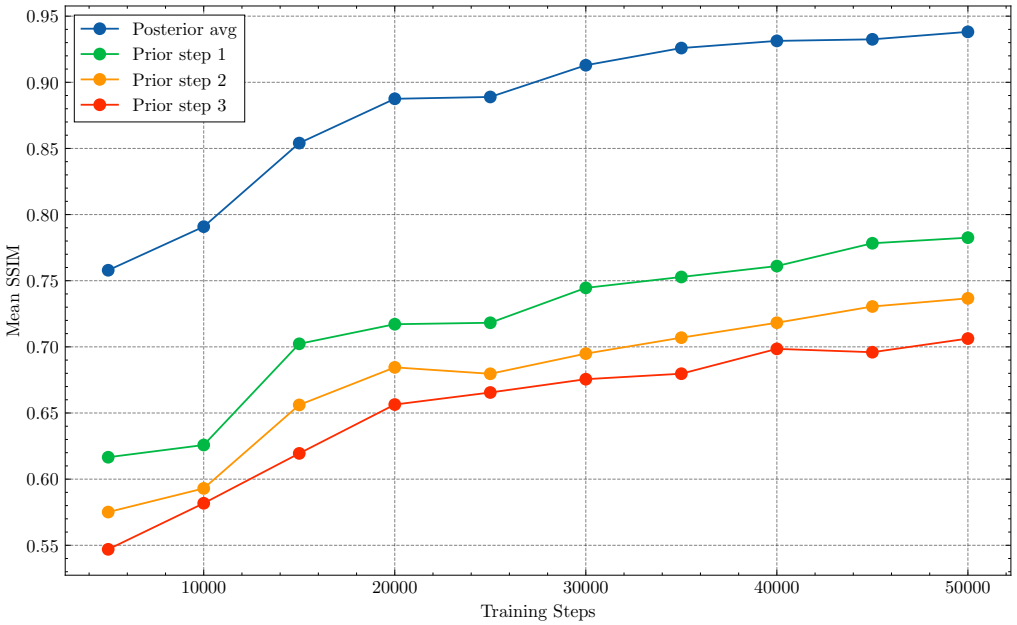


Figure 5.1.: The average quality evolution of the reconstructed prior and posterior states evolves continuously during training, shown by the increasing SSIM values.

in PlaNet-ClothPick); (ii) use of data augmentation during training; and (iii) Transformer depth (four vs. two layers). Table 5.2 lists the configurations, which were implemented as part of this thesis to systematically test the effect of each choice. The evaluation set follows VSF [43] and PlaNet-ClothPick, using 40 selected trajectories with diverse initial garment coverage.

The following sections present training results, reconstruction quality, reward prediction accuracy, and evaluation of the quality of the action selected by the planning algorithm.

5.2. Training Dynamics

This section tracks how predictions from the adapted TSSM evolve during training, using reconstruction metrics and loss curves. Analysing training dynamics provides insight into model convergence and stability, which are prerequisites for robust planning.

Figure 5.1 shows that posterior and prior reconstructions steadily improve, plateauing around 25,000 steps, coinciding with the end of learning rate decay, until reaching another plateau towards the end of training.

Figure 5.2 shows the loss curves during training for both stochastic distributions. The KL divergence loss is much lower for categorical distribution models, which indicates that this configuration prioritises the training of a good prior prediction, based on the previous posterior state. They also show higher image loss, suggesting weaker reconstructions.

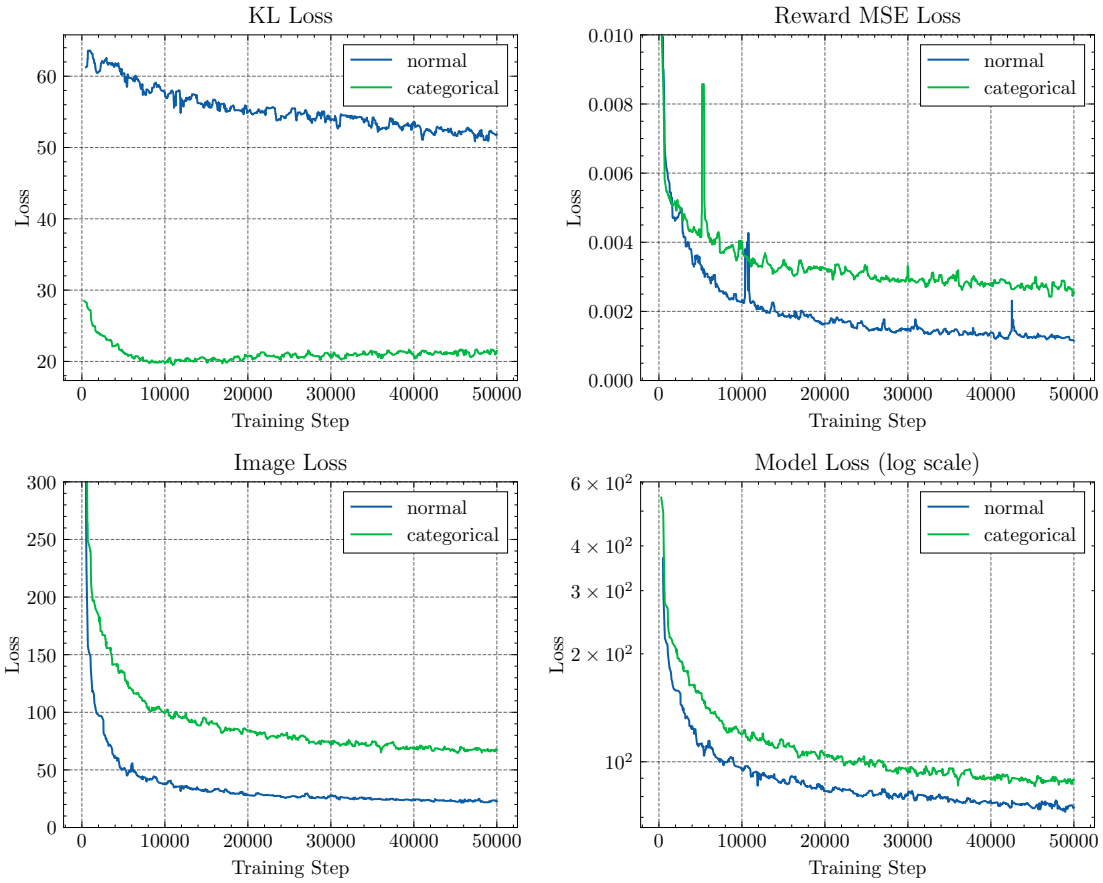


Figure 5.2.: Loss curves for TSSM models using normal and categorical stochastic state distributions. The image loss drops sharply in the early training phase, indicating fast learning of static visual features, while the KL loss decreases more slowly and becomes more influential in later stages as reconstruction loss plateaus. This shift in the effective loss balance leads to a greater focus on refining latent dynamics rather than further reducing pixel-level error. Normal-distribution models consistently achieve lower image, reward, and overall model losses compared to categorical models, supporting their superior reconstruction and prediction performance in the CDO domain.

Table 5.3.: MSE and SSIM values for Posterior and Prior (steps 1–3)

Model	Posterior		Prior1		Prior2		Prior3	
	MSE ↓	SSIM ↑	MSE ↓	SSIM ↑	MSE ↓	SSIM ↑	MSE ↓	SSIM ↑
rssm	0.58	87.79	2.40	72.37	3.04	69.46	3.73	66.51
td-05-07	**0.16**	93.42	1.42	79.61	1.80	76.14	2.04	73.02
td-07-07	0.60	84.94	1.36	78.58	1.81	74.99	2.09	72.43
td-21-07	0.17	92.78	**1.19**	**80.32**	**1.69**	**76.38**	**1.99**	**73.52**
td-23-07	0.50	85.29	1.54	76.39	1.89	72.82	2.10	70.48
td-01-08	0.17	**93.56**	1.63	76.87	2.40	72.18	2.84	69.54

5.3. Reconstruction Quality Evaluation

Reconstruction quality reflects how accurately the model simulates observed states. Posterior quality indicates the learning of accurate latent representation; prior quality indicates multi-step prediction capability and the capturing of the environment dynamics. Both are essential for effective model-predictive control (MPC) planning, as goal-based MPC uses the predicted state reconstructions to choose an action. Although the TSSM decoder outputs a probability distribution over pixel intensities, the mean prediction can be extracted and compared to ground truth images, allowing the use of mean squared error (MSE) for direct comparison with deterministic decoders. A common measure to assess reconstruction quality is to compare images with ground truth by calculating the MSE between pixels. Wang *et al.* [57] also propose the structured similarity index (SSIM), which aligns more closely with the differences perceived by the human eye. Two reasons justify the use of SSIM alongside MSE: First, blurry predictions of latent dynamics models (LDMs) are mentioned as potential sources of poor CDO manipulation performance, because they directly influence the quality of planning around the edges of the fabric [9], [41], [43]. Blurriness represents a specific type of visual degradation that SSIM is well suited to assess, and can remain undetected in the MSE results [57]. Second, the SSIM allows for focused evaluation on the areas of images containing the garment itself, rather than being disproportionately influenced by the uniform background areas [57].

Table 5.3 presents the MSE and SSIM across reconstructed posterior and prior predictions over three rollout steps. Posterior reconstructions produce consistently low MSE and high SSIM, reflecting the availability of true observations to generate the latent state. Prior predictions degrade over time, highlighting the accumulation of errors in predictions over longer horizons. The 1-step prior reflects the model's internal ability to predict future observations solely from latent rollouts, and is therefore a sensitive indicator of planning reliability. All adapted TSSM variants in Table 5.2 achieve better first-step prior quality than PlaNet-ClothPick, with MSE reduced by 50% and SSIM improved by close to 6 points for models using a normal distribution, suggesting that Transformer-based LDMs better capture complex temporal dependencies.

5.4. Reward Prediction Accuracy

This section evaluates reward predictions by computing the MSE between predicted and true scalar rewards for each trajectory. Reward-based MPC directly uses the predicted rewards to assess the quality of sampled actions, and is common in CDO manipulation. Three reward model configurations from Chapter 3 were evaluated. For comparability, MSE is used for probabilistic reward models taking the mean of the predicted distribution, ensuring that all models are evaluated under the same metric.

Table 5.4.: Reward MSE values for Posterior and Prior (steps 1-3). Italic values in the third column were expected to be worse.

Model	Posterior	Prior 1	Prior 2	Prior 3
planet-clothpick	0.28	0.79	0.90	*0.66*
td-05-07	0.23	0.67	0.92	**0.83**
td-07-07	0.62	**0.63**	0.86	0.95
td-21-07	**0.20**	**0.63**	**0.76**	0.96
td-23-07	0.58	0.69	0.77	*0.43*
td-01-08	0.25	0.81	1.00	1.23

Table 5.4 reports the MSE between predicted and true rewards across posterior and prior steps. The model `td-21-07` performs best, aligning with the reconstruction accuracy results in Table 5.3. The rewards are predicted from the same latent state as the reconstructions, using a different neural model, so an alignment of reward prediction quality and reconstruction quality indicates that both models are well trained.

Models using categorical distributions (`td-07-07`, `td-23-07`) predict prior rewards well but perform worse on posterior predictions. Some models show nonmonotonic error patterns (e.g. lower error at Prior 3 than Prior 2), requiring further analysis.

5.5. Ablation Studies

This section analyses two sets of ablations defined in Chapter 3. First, the effect of data augmentation and stochastic state distribution type, using models `td-21-07` (normal) and `td-23-07` (categorical) from Table 5.2. Second, the effect of different reward model configurations, evaluated on the best-performing TSSM variant.

Figure 5.3 compares stochastic distributions and data augmentation effects on reconstruction performance. Understanding these factors helps optimise architecture design choices that directly impact predictive quality and planning success. TransDreamer originally used a discrete categorical distribution to model the stochastic state, and PlaNet-ClothPick uses a continuous normal distribution. Normal distributions outperform categorical ones, improving posterior RMSE by 40%. It also shows that data augmentation further improves prior prediction, regardless of distribution, as this generally improves data efficiency and the sim-to-real gap by helping expose the agent to more randomised observations [4].

The reward model ablation uses the best-performing TSSM variant `td-21-07` to compare the three reward model configurations described in Chapter 3. Modelling the reward as a probability distribution was originally done in TransDreamer and Figure 5.4 shows that probabilistic reward modelling, though worse on the first step, performs better at predicting longer horizons, which can improve planning [13]. The RSSM reward model used in PlaNet-ClothPick is significantly smaller than the multi-layer perceptron (MLP) used in TransDreamer, and showed comparable performance for the first step prior, but performance degradation for step two and three, highlighting its limited robustness.

5.6. Evaluation of TSSM in Simulation

This section presents the first closed-loop simulation-based evaluation of a TSSM agent for CDO manipulation, using the best-performing model (`td-21-07`) integrated into Agent-

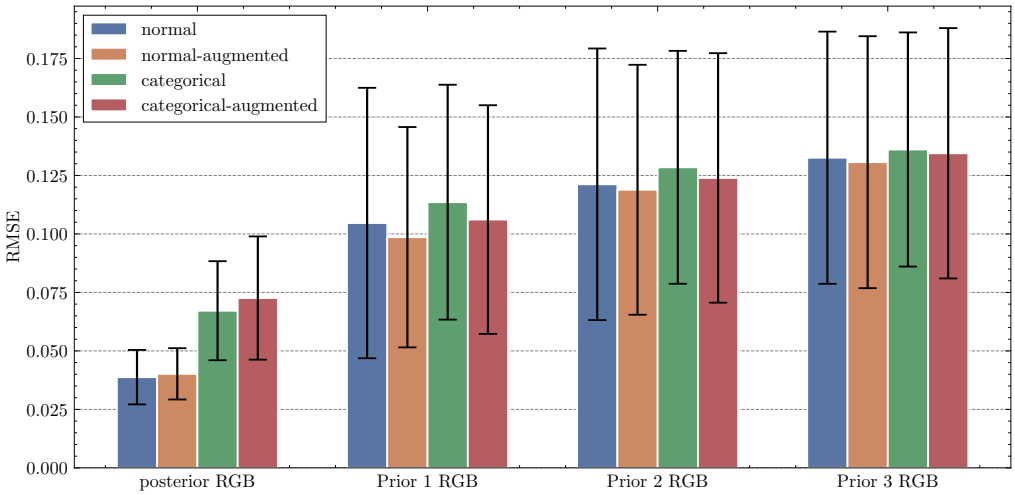


Figure 5.3.: The RMSE of reconstructions across different model configurations shows the effect of data augmentation and the stochastic distribution chosen to model the uncertainty in the latent state.

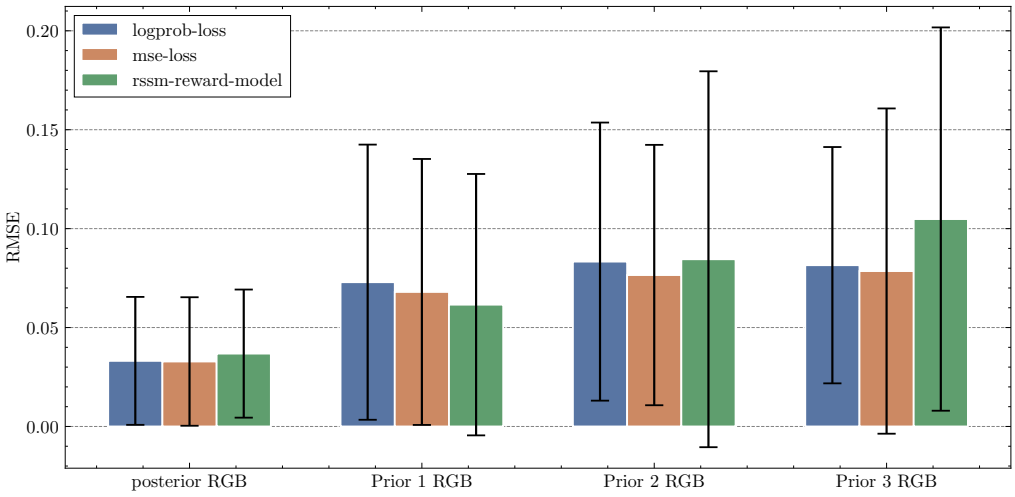


Figure 5.4.: The RMSE reward comparison across different reward models highlight the capability for long term predictions of using a probabilistic reward (`logpro-loss`), and the degrading performance of the smaller RSSM reward model.

Table 5.5.: Average reward and normalised coverage metrics across different models.

Model	Avg Reward	Avg NC	Avg Planning Time
planet-clothpick	0.0197	0.445	0.07768
td-05-07	0.0164	0.596	0.12021
td-21-07	0.0174	0.545	0.11979

Arena via the custom `act()` implementation from Chap. 4. Here, "closed-loop" refers to running the MPC planner over short, fixed-length simulated trajectories loaded from the test dataset, where the environment state is updated after each predicted action, but no live physics simulation is performed. The goal is to assess whether the selected actions lead to increased garment coverage, as measured by normalised coverage (NC), defined in Equation 5.1. This follows the principle that flattening tasks should increase the area covered by the cloth with each step. NC refers to the total area covered by the cloth (s) after an action taken by the agent, scaled by the maximum possible area covered by the cloth in the flattened state, denoted as s_{max} [2].

$$NC = \frac{s}{s_{max}} \tag{5.1}$$

To evaluate the planning performance in this closed-loop setting, fixed recorded trajectories from the test set are used to initialise the simulated environment. The MPC then uses the implemented cost function to select the best action from a sampled subset of possible actions. After each action is chosen, the environment state is updated using the trajectory observations, and the process repeats for the defined horizon. This approach allows consistent, reproducible comparisons across models, though it does not capture the full variability of a live physics simulation. To assess the average area after unrolling predicted actions of the first 10 states of the test trajectories used throughout this chapter, I use Equation 5.1 to calculate the coverage compared to the goal state of each sequence. Table 5.5 shows that the TSSM-based models surpass PlaNet-ClothPick in the normalised coverage, indicating the actions selected by the MPC planning on a TSSM achieve greater flattening in each step. While absolute reward values are similar for both models, NC offers a clearer picture of physical task success.

Collectively, these metrics demonstrate that the adapted TSSM can successfully plan in the CDO domain when embedded within Agent-Arena, a capability not previously reported. The next chapter discusses the broader implications of these results, which outlines directions for future research.

6. Discussion

This chapter reflects on the results presented in this thesis. By examining both quantitative and qualitative results, I assess the effectiveness of the proposed Transformer state-space model (TSSM) architecture in capturing the complex dynamics of cloth-like deformable objects (CDOs), its planning performance in model-predictive settings, and the impact of architectural and stochastic design choices. This discussion aims to place the results within the broader literature on model-based reinforcement learning (MBRL) for CDO manipulation. The analysis emphasises that this is the first evaluation of a TSSM in the CDO domain, allowing direct comparison with the established recurrent state-space model (RSSM) baseline in both predictive accuracy and planning performance.

6.1. Latent State Quality

The reconstruction and reward prediction results (Section 5.2–5.4) show that the adapted TSSM learns a higher-quality latent representation of cloth dynamics than the RSSM baseline. Posterior accuracy and prior rollout fidelity are both improved, indicating that the model not only encodes the current state effectively but also predicts plausible future states over multiple steps — a critical property for model-predictive control (MPC) planning.

Similar results in Table 5.4 underline the superior performance of the TSSM in learning the complex dynamics during the training process and its capability to accurately unroll future states given an action, which is essential for MPC planning, both reward and goal based. In latent dynamics theory, improvements in reconstruction quality are associated with reduced compounding error during long-horizon planning, which may partly explain the observed performance gains in normalised coverage (NC) when using TSSM [2]. The models trained with a categorical stochastic distribution were not able to learn the same representation quality, which is backed up by the loss curves in Section 5.2. For models using a categorical distribution (`td-07-07`, `td-23-07`), training tended to prioritise KL divergence minimisation over accurate latent state modelling, resulting in priors that closely match the posterior but do not improve actual prediction quality. This suggests that loss-balancing strategies could be a confounding factor in performance differences. This is evident from models `td-07-07` and `td-23-07` in Table 5.3 and 5.4.

Reducing model depth from four to two layers had little effect on posterior quality (image-loss-driven) but degraded prior predictions by $\sim 4\%$ and reward prediction accuracy by $\sim 20\%$, suggesting that sufficient capacity is required to capture the high-dimensional dynamics of cloth. This indicates that the size of the model is important for capturing essential information. Future work is needed to assess this effect in more detail, as Zhang *et al.* [12] found that increasing the model size from 4 to 6 layers did not yield additional performance gains in the Atari benchmark, and Plaat *et al.* [25] argues that the model size depends on the specific task, data, and hardware. Given that cloth manipulation involves high-dimensional continuous deformations, it is plausible that model depth requirements differ from domains like Atari, and thus architectural scaling studies should be replicated under similar physical dynamics to confirm transferability.

Visual rollouts provide qualitative insights into the latent states and help validate the quantitative metrics. As shown in Figure 6.1, the TSSM produces sharper contours, both

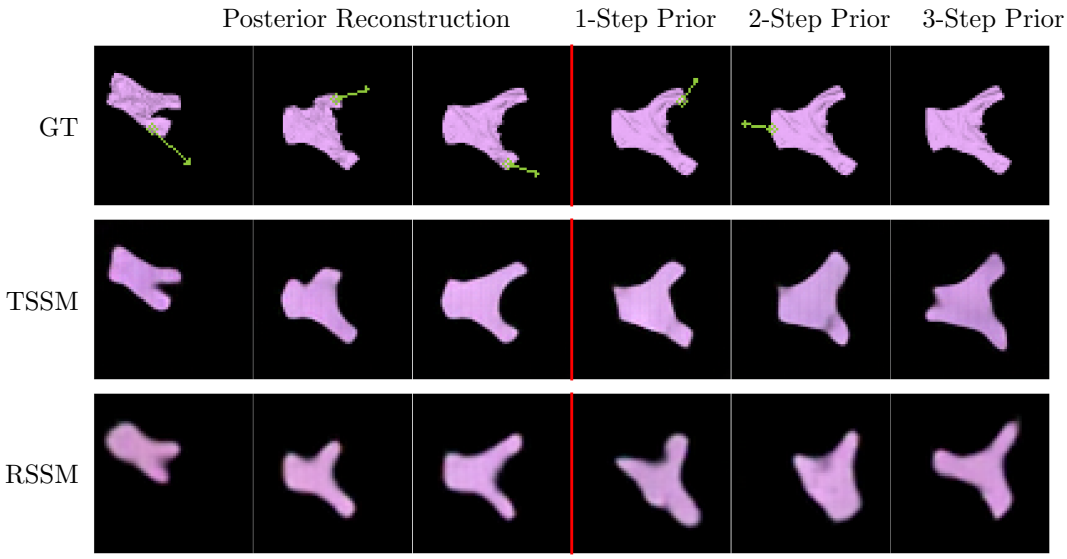


Figure 6.1.: Predicted cloth manipulation rollouts for a long-sleeve shirt. The top row (GT) shows the ground-truth sequence where green arrows indicate the pick-and-place actions applied to the garment — the arrow starts at the "pick" point and ends at the "place" location. The first three columns show posterior reconstructions (model predictions when given the real images), while the last three columns show prior predictions (purely simulated rollouts) for 1-step, 2-step, and 3-step ahead forecasts using the GT actions. The TSSM model (middle row) produces sharper garment boundaries and preserves the shirt's shape better over multiple steps compared to the RSSM model (bottom row), whose predictions appear more blurred and distorted over time.

in posterior and prior reconstruction quality, which aligns with the findings in Table 5.3. Kadi *et al.* [9] attribute blurry predictions as one of the main issues for planning in the latent space and accurate pick actions. This visual sharpness likely translates into more precise grasp-point proposals during MPC rollouts, reducing the probability of wasted actions and leading to downstream efficiency gains.

Figure 6.2 shows that most reconstruction gains occur before 25k steps, with sharper contours and more accurate garment shape emerging early. Loss curves in Figure 5.2 confirm this pattern: image reconstruction loss drops steeply in the first phase, indicating rapid convergence of posterior quality, while KL loss decreases more gradually. As image loss plateaus, the fixed loss weighting shifts optimisation toward KL regularisation, steadily improving the alignment between prior and posterior latent states. This sustained KL-driven refinement corresponds to the more stable and physically plausible multi-step priors seen at 50k.

6.2. Stochastic Distribution

This section aims to investigate the performance differences between the stochastic distribution choices both for latent states and rewards. Although discrete distributions have performed well on different Atari benchmarks [5], [6], [8], the performance is not transferable to the CDO domain. One likely reason for this domain gap is the different temporal and spatial correlation structures in cloth deformation compared to Atari frame sequences, which may make fixed-bin discretisation overly coarse.

Prediction entropy in this context measures the uncertainty of the latent state distribution during rollouts. For posterior states, lower entropy over training indicates that the model

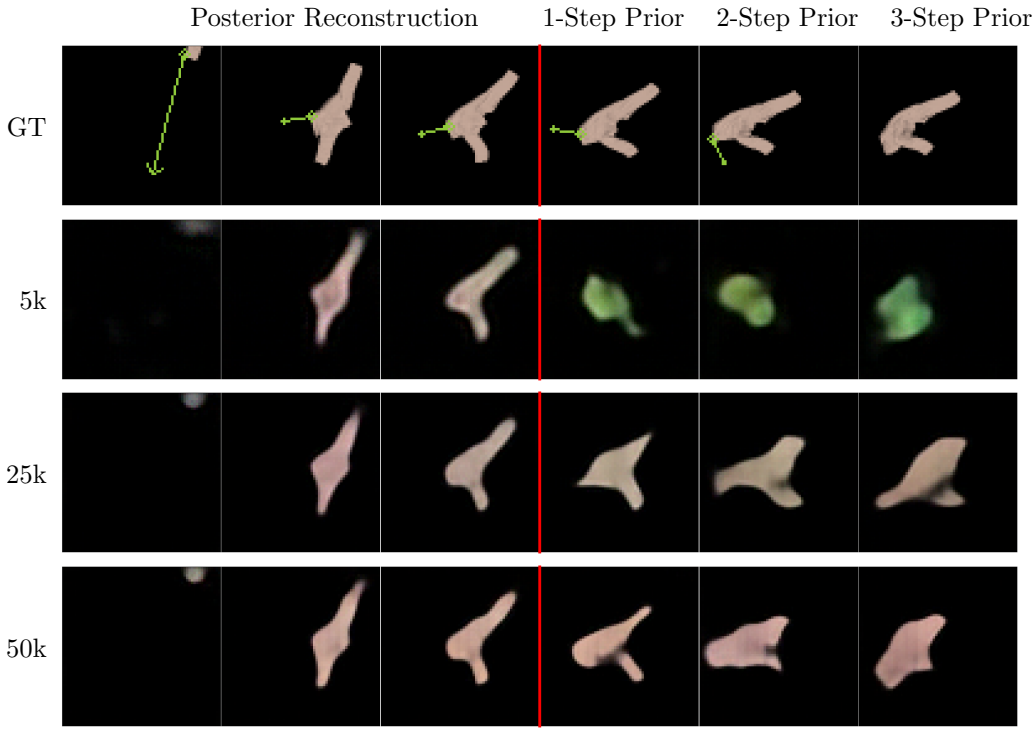


Figure 6.2.: Rollout quality of the TSSM at different training checkpoints (GT = ground truth sequence; 5k/25k/50k = number of training steps). The largest gains occur between 5k and 25k steps, with sharper contours and more consistent garment shape in multi-step priors. Loss curves show that posterior reconstructions (image loss) converge rapidly, while the slower decline in KL loss indicates continued alignment of prior and posterior latent states in later stages. This sustained KL improvement corresponds to the more stable and physically plausible multi-step priors seen at 50k.

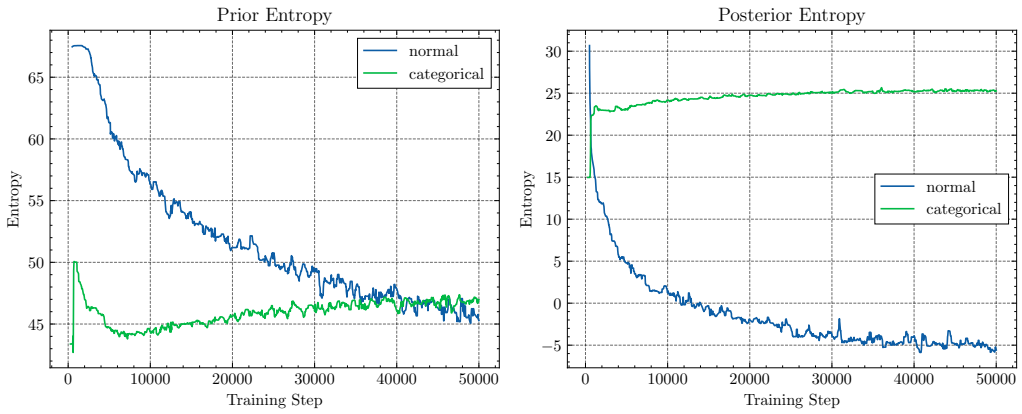


Figure 6.3.: Entropy of TSSM predictions during training for normal and categorical latent distributions. Posterior entropy reflects uncertainty in state estimation, while prior entropy reflects multi-step prediction stability. The normal model's entropy decreases steadily, showing increasingly confident and stable latent inferences, whereas the categorical model retains high posterior entropy throughout, indicating persistent uncertainty and reduced suitability for modelling continuous cloth deformations.

is becoming more confident in its state estimates. For priors, it reflects the stability of multi-step predictions without direct observation. In long-horizon planning, lower and more stable entropy is generally desirable, as it means the model can commit to confident rollouts without compounding uncertainty.

Figure 6.3 shows a stark contrast between the normal and categorical variants. The categorical variant maintains high posterior entropy throughout the training, whereas the normal model converges to a lower entropy, suggesting it learns more confident and stable latent inferences. One possible explanation is the limited representational capacity of fixed categorical bins, because they may struggle to represent the subtle, continuous changes in fabric configuration, leading to a loss of critical information necessary for effective manipulation. This is consistent with the reconstruction and planning results reported earlier, where categorical models underperformed. This highlights a key limitation in the transfer of discrete latent spaces from Atari domains to high-dimensional continuous deformation tasks such as CDO manipulation. Future work could explore these results further to analyse the practical differences between different distributional choices.

6.3. Planning Performance

Using the best-performing TSSM variants (`td-21-07` and `td-05-07`) within the AGENT-ARENA, using the ClothPick-MPC algorithm of PLANET-CLOTHPICK marks the first demonstration of Transformer-based planning in a fabric flattening task. The evaluation was conducted in a closed-loop simulation using fixed-length recorded trajectories from the test set, where the MPC algorithm predicted and executed actions sequentially within the pre-recorded sequence. While this provides a controlled comparison between models, it does not replicate the dynamics of a live, continuously simulated environment. While the average reward values across these models appear numerically similar, the NC metric offers a more direct and physically intuitive measure of task success, indicating that actions selected by the MPC planner using a TSSM achieve greater fabric flattening in each step. This supports the earlier finding in Section 6.1 that more accurate latent states directly improve the quality of imagined trajectories used by the planner.

Previous research identified that RSSMs similar to the foundational PlaNet struggled with the flattening of the fabric due to an inherent difficulty in accurately modelling the sharp discontinuities in the contour of the cloth, which led the MPC planner to attempt to pick actions slightly outside the fabric [9]. PlaNet-ClothPick overcame these issues by adopting and refining the VSF reward function, and adopting KL balancing to improve the accuracy of latent prediction. The higher NC values achieved with TSSM-based planning suggest that the architecture's improved latent predictions directly translate into more effective action sequences for fabric flattening. It is also possible that differences in training data augmentation or subtle changes in reward shaping contributed to these gains, warranting controlled ablation studies. This aligns with the results discussed in Section 6.2, that discrete latent space representations can struggle with the continuous and high-dimensional properties of cloth manipulation. The TSSM models likely provide more confident and stable latent inferences, enabling the MPC to generate action sequences that lead to more effective physical changes in the fabric's configuration, even if the average absolute reward remains similar. Although the TSSM models exhibit a slightly longer average planning time compared to RSSM (approximately 0.12s vs 0.078s), this trade-off is acceptable given the significant gains in physical flattening performance. This outcome highlights an interesting direction for future advancements beyond RSSM-based sequence modelling. To fully assess robustness and generalisation, however, the next step should be to extend beyond fixed closed-loop simulations to longer-horizon, interactive simulations, and ultimately real-world trials.

6.4. Evaluation of Objectives

This work achieved all primary and secondary objectives: it replaced the recurrent neural network (RNN)-based backbone of PlaNet-ClothPick with an adapted Transformer state-space model (TSSM), integrated this into Agent-Arena, trained the new model and conducted the first evaluation of such a model in the challenging cloth-like deformable object (CDO) domain. Comparative experiments showed consistent TSSM outperformance over RSSM in both reconstructed state accuracy and reward prediction. These results improved the policy generated by the ClothPick-MPC planning algorithm in simulated fabric-flattening tasks compared to RSSM, which was measured by the normalised coverage (NC).

These results validate that the proposed Transformer integration addressed the identified limitations of PlaNet-ClothPick, providing both improved predictive quality and downstream planning performance in the challenging domain of cloth manipulation. While results strongly support the Transformer's advantages in modelling cloth dynamics, more extensive simulation, beyond the action predictions based on a fixed simulated trajectory, and subsequent real-world testing will be necessary to confirm robustness under sensor noise and actuation variability.

7. Conclusion

This thesis presents the first adaptation of a Transformer state-space model (TSSM) to the domain of cloth manipulation under the model-based reinforcement learning (MBRL) framework. Focusing on the canonical fabric flattening task, the work extends PlaNet-ClothPick by replacing its recurrent state-space model (RSSM) with a Transformer-based alternative, while keeping all other components, including the model-predictive control (MPC) planner, identical. This integration, implemented within the open-source framework Agent-Arena, allows a direct and controlled comparison of the two architectures and provides a reusable platform to evaluate Transformer-based latent dynamics models (LDMs) in the manipulation of deformable objects. The subsequent investigation evaluated how well the new TSSM performs relative to the recurrent baseline.

The study began by identifying the limitations of the RSSM in cloth-like deformable object (CDO) manipulation: low data efficiency and blurry prediction reconstructions that degrade further over multi-step rollouts. These weaknesses arise from the limited memory capacity of the recurrent backbone to capture the complex characteristics of cloth dynamics. This causes a loss of fine spatial details in state predictions. In contrast, the proposed TSSM leverages self-attention to process all past latent states and actions directly, preserving richer temporal context and sharper reconstructions. The expectation was that sharper imagined reconstructions and more accurate reward predictions would translate into more reliable MPC planning. To evaluate this, both RSSM and TSSM were trained on the same dataset of simulated cloth interactions. Performance was measured across three dimensions: (i) reconstruction quality from imagined latent states, (ii) accuracy of reward prediction from imagined states, and (iii) overall task performance in simulated fabric flattening as measured by coverage-based metrics.

Key Findings

The results of this first TSSM–cloth manipulation study in MBRL show that TSSM outperforms RSSM in key metrics. First, in reconstruction quality, the TSSM produced sharper and more consistent cloth boundaries over multiple imagined steps. This improvement was evident in both quantitative metrics (lower mean squared error (MSE) and higher structured similarity index (SSIM)) and qualitative rollouts. While the RSSM reconstructions tended to lack fine detail, the TSSM maintained sharper contours (Figure 6.1). Second, in reward prediction, the TSSM achieved consistently lower error across multiple rollout lengths. This indicates that it maintained a more accurate feature representation in its latent space, enabling MPC to select action sequences more reliably. Third, in planning performance, the TSSM-based agent achieved higher average coverage in the simulated action predictions. These results confirm that the adapted TSSM not only addresses the identified RSSM limitations but also advances the state of the art in cloth manipulation under MBRL.

The performance improvements of the TSSM are consistent with the strengths of Transformer architectures observed in other domains. The sharper reconstructions suggest that the TSSM captures spatially detailed information over longer horizons. Better reward predictions indicate that the latent representations of the new model preserve the characteristics that matter for decision-making. Another notable observation is that the TSSM

achieved these gains without changing the planner or the reward function. This isolates the improvement to the Transformers internal ability to represent and predict cloth dynamics, supporting the conclusion that the choice of backbone architecture has a direct and significant impact on downstream task performance in MBRL.

Limitations and Future Work

While the TSSM-based models showed significant improvements over traditional RSSM architectures in cloth manipulation tasks, the following limitations highlight opportunities for future work. These constraints are not only technical but also affect how the present findings can be generalised beyond the specific simulation and task setup used in this thesis.

First, the experiments were conducted entirely in closed-loop simulation, using the dataset generated under controlled conditions. In line with the defined objectives, planning performance was assessed in closed-loop simulation using fixed offline trajectories, which limits the ability to draw direct conclusions about real-time interaction robustness. Although simulation allows for large-scale data collection and precise evaluation, the sim-to-real gap remains a challenge in cloth manipulation. Physical fabrics can exhibit properties such as varying stiffness, friction, or small-scale wrinkles, which could affect generalisation capabilities of the agent. Second, the Transformer architecture comes with a higher computational cost compared to the recurrent baseline. Although inference with 20 step sequences is tractable, scaling to very long sequences or deploying on resource-constrained hardware could prove difficult. This limitation may be particularly relevant for embedded control on mobile robots or assistive devices, where computational budgets and power consumption are tightly constrained.

Addressing these limitations motivates several directions for future work, each aimed at improving robustness, efficiency, and applicability in real-world cloth manipulation. This work used offline reinforcement rearning (RL). Extending experiments to more diverse and dynamic online RL environments, such as those provided by the real-time cloth simulation capabilities of SoftGym, would allow the model to encounter a broader spectrum of fabric configurations. Incorporating stochastic exploration and dynamic data collection in such environments could further enhance learning robustness. Diversity in simulated experiences can also act as a proxy for domain randomisation, potentially narrowing the sim-to-real gap without additional real-world data collection. A next step is to evaluate the generalisability of the TSSM and robustness in physical robotic systems. Transferring latent dynamics models trained in simulation to the real world requires addressing the sim-to-real gap, including sensor noise, actuation delays, and visual domain shift. A staged approach, beginning with controlled lab setups and gradually introducing unstructured environments, could help isolate failure modes before full real-world deployment. Techniques such as domain randomisation and visual pre-training may help facilitate this transition. The current implementation focuses on short-horizon planning using a single step MPC. Future research may investigate improvements to multi-step planning within the MPC framework, potentially adopting multi-step losses for training, as Burchi *et al.* [13] find this increases performance in Atari environments. The increased computational load of multi-step MPC in high-dimensional latent spaces should be carefully weighed against any performance gains, especially for real-time applications. While reducing the number of Transformer layers showed performance trade-offs, a more systematic evaluation of architectural scaling laws for TSSM in deformable object domains could provide more insights into optimal model configurations. Zhang *et al.* [12] found that larger Transformers perform worse in some Atari environments, which may prove valuable for CDO

manipulation. A systematic evaluation of layer depth, attention head count, and embedding size, paired with controlled ablation studies, could help map a more precise scaling law for deformable object tasks.

Concluding Remarks

This thesis has shown that replacing a recurrent backbone with a Transformer-based one in a latent dynamics model (LDM) can improve prediction quality and planning performance for cloth manipulation in model-based reinforcement learning (MBRL) for the first time in the cloth-like deformable object (CDO) domain. Trained for fabric flattening, the Transformer state-space model (TSSM) outperformed the recurrent state-space model (RSSM) in reconstruction quality, reward prediction accuracy, and action selection. These gains come from the Transformers ability to maintain detailed, task-relevant information over long horizons, a property that aligns well with the temporal complexity of cloth dynamics. While challenges remain in terms of generalisation to the real world and extension to other tasks, the results suggest that Transformer-based LDMs hold strong promise for advancing the state of the art in robotic manipulation of deformable objects.

In answering the central research question of whether a Transformer-based LDM can improve latent state prediction and planning performance in cloth manipulation, the evidence from this work shows that the adapted TSSM exceeded the capabilities of the RSSM baseline. This represents both a methodological advance and a practical, reusable integration for future model-based control research in complex deformable object manipulation.

A. Ethics

Training and evaluation are performed purely on synthetic data, hence the appropriate *CS Artefact Evaluation Form.*

UNIVERSITY OF ST ANDREWS
TEACHING AND RESEARCH ETHICS COMMITTEE (UTREC)
SCHOOL OF COMPUTER SCIENCE
PRELIMINARY ETHICS SELF-ASSESSMENT FORM

This Preliminary Ethics Self-Assessment Form is to be conducted by the researcher, and completed in conjunction with the Guidelines for Ethical Research Practice. All staff and students of the School of Computer Science must complete it prior to commencing research.

This Form will act as a formal record of your ethical considerations.
Tick one box

- ☐ **Staff Project**
- ☒ **Postgraduate Project**
- ☐ **Undergraduate Project**

Title of project

Towards Transformer-Driven Latent Dynamics in Cloth Manipulation

Name of researcher(s)

Florian Pfleiderer

Name of supervisor (for student research)

Dr. Kasim Terzic

OVERALL ASSESSMENT (to be signed after questions, overleaf, have been completed)

Self audit has been conducted **YES** ☒ **NO** ☐

There are no ethical issues raised by this project

Signature Student or Researcher

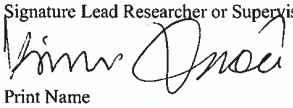
Print Name

Florian Pfleiderer

Date

04/04/2025

Signature Lead Researcher or Supervisor



Print Name

Dr. Kasim Terzic

Date

04/04/2025

This form must be date stamped and held in the files of the Lead Researcher or Supervisor. If fieldwork is required, a copy must also be lodged with appropriate Risk Assessment forms. The School Ethics Committee will be responsible for monitoring assessments.

B. Objectives

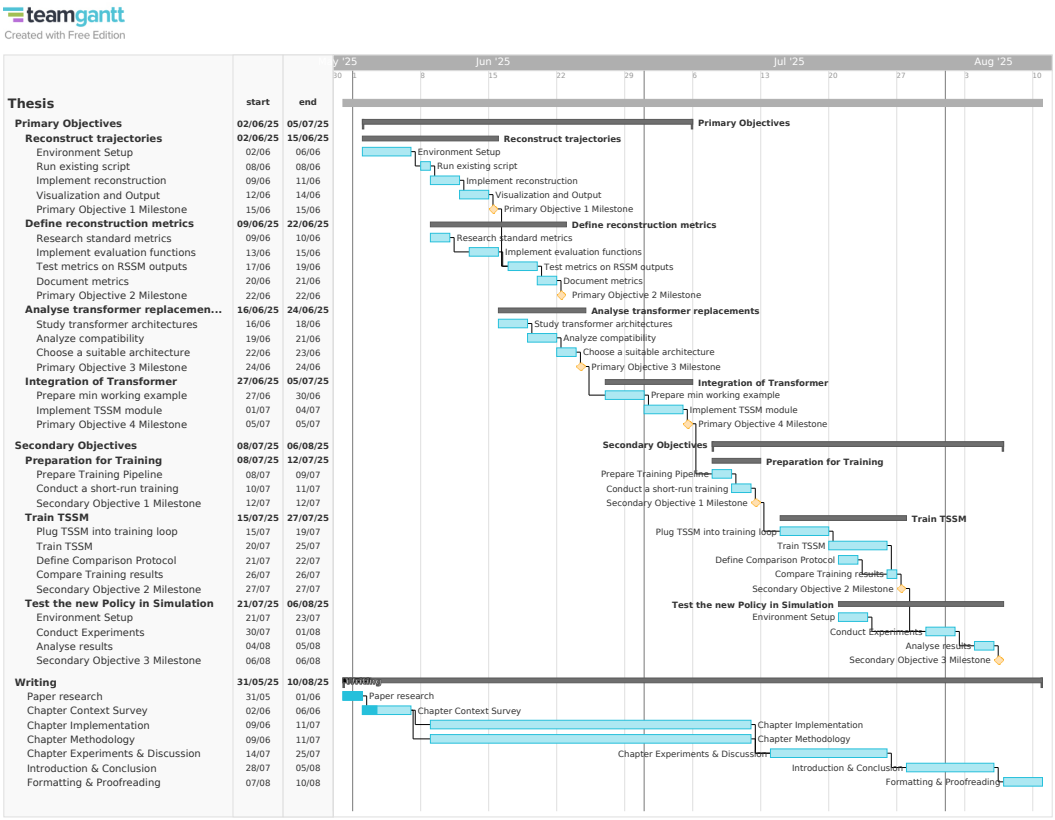
Primary Objectives

1. Develop a script to reconstruct prior, posterior, and reward trajectories from a test dataset using the given RSSM-based latent dynamic model trained on a synthetic garment dataset.
2. Define quantitative metrics to assess the quality of the reconstructed predictions and the latent representations.
3. Investigate strategies to replace the recurrent neural network (RNN) with a transformer by analysing compatibility with the PlaNet latent prior prediction mechanism.
4. Integrate the selected transformer module into the RSSM architecture.

Secondary Objectives

1. Ensure that the new latent dynamic model can be trained and evaluated with the given garment dataset generated by pick-and-place policies.
2. Train and evaluate the transformer-based implementation and compare the latent representation quality with the original RNN-based PlaNet-ClothPick.
3. Test the policy generated by the model-predictive control (MPC) planning algorithm using the proposed new latent dynamic model in simulation, and compare its performance against PlaNet-ClothPick's.

C. Workplan


teamgantt
Created with Free Edition
y '25
Jun '25
Jul '25
Aug '25
30 1 8 15 22 29 6 13 20 27 3 10
Thesis start end
Primary Objectives 02/06/25 05/07/25
Reconstruct trajectories 02/06/25 15/06/25
Environment Setup 02/06 06/06
Run existing script 08/06 08/06
Implement reconstruction 09/06 11/06
Visualization and Output 12/06 14/06
Primary Objective 1 Milestone 15/06 15/06
Define reconstruction metrics 09/06/25 22/06/25
Research standard metrics 09/06 10/06
Implement evaluation functions 13/06 15/06
Test metrics on RSSM outputs 17/06 19/06
Document metrics 20/06 21/06
Primary Objective 2 Milestone 22/06 22/06
Analyse transformer replacemen... 16/06/25 24/06/25
Study transformer architectures 16/06 18/06
Analyze compatibility 19/06 21/06
Choose a suitable architecture 22/06 23/06
Primary Objective 3 Milestone 24/06 24/06
Integration of Transformer 27/06/25 05/07/25
Prepare min working example 27/06 30/06
Implement TSSM module 01/07 04/07
Primary Objective 4 Milestone 05/07 05/07
Secondary Objectives 08/07/25 06/08/25
Preparation for Training 08/07/25 12/07/25
Prepare Training Pipeline 08/07 09/07
Conduct a short-run training 10/07 11/07
Secondary Objective 1 Milestone 12/07 12/07
Train TSSM 15/07/25 27/07/25
Plug TSSM into training loop 15/07 19/07
Train TSSM 20/07 25/07
Define Comparison Protocol 21/07 22/07
Compare Training results 26/07 26/07
Secondary Objective 2 Milestone 27/07 27/07
Test the new Policy in Simulation 21/07/25 06/08/25
Environment Setup 21/07 23/07
Conduct Experiments 30/07 01/08
Analyse results 04/08 05/08
Secondary Objective 3 Milestone 06/08 06/08
Writing 31/05/25 10/08/25
Paper research 31/05 01/06
Chapter Context Survey 02/06 06/06
Chapter Implementation 09/06 11/07
Chapter Methodology 09/06 11/07
Chapter Experiments & Discussion 14/07 25/07
Introduction & Conclusion 28/07 05/08
Formatting & Proofreading 07/08 10/08
Primary Objectives
Reconstruct trajectories
Environment Setup
Run existing script
Implement reconstruction
Visualization and Output
Primary Objective 1 Milestone
Define reconstruction metrics
Research standard metrics
Implement evaluation functions
Test metrics on RSSM outputs
Document metrics
Primary Objective 2 Milestone
Analyse transformer replacements
Study transformer architectures
Analyze compatibility
Choose a suitable architecture
Primary Objective 3 Milestone
Integration of Transformer
Prepare min working example
Implement TSSM module
Primary Objective 4 Milestone
Secondary Objectives
Preparation for Training
Prepare Training Pipeline
Conduct a short-run training
Secondary Objective 1 Milestone
Train TSSM
Plug TSSM into training loop
Train TSSM
Define Comparison Protocol
Compare Training results
Secondary Objective 2 Milestone
Test the new Policy in Simulation
Environment Setup
Conduct Experiments
Analyse results
Secondary Objective 3 Milestone
Writing
Paper research
Chapter Context Survey
Chapter Implementation
Chapter Methodology
Chapter Experiments & Discussion
Introduction & Conclusion
Formatting & Proofreading

D. Environment Setup

This project runs on a lab machine equipped with a multi-core CPU, 32GB RAM, 500GB storage, and an NVIDIA RTX 3060 GPU (12GB). The GPU driver version is 530.41.03 with CUDA 12.1 on the host OS. GUI applications are forwarded via XQuartz and SSH X11.

Dependencies

Install XQuartz (e.g. if on macOS) and run:

```
xhost +
```

SSH into the server using trusted X11 forwarding:

`ssh -Y your_user@gpu-pc-address`

Ensure `DISPLAY` and `.Xauthority` are set:

```
echo $DISPLAY
ls ~/.Xauthority
```

Make sure openGL works (a window with spinning gears should pop up):

```
glxgears
```

Make sure the following dependencies are installed on the Ubuntu machine:

```
build-essential libgl1-mesa-dev freeglut3-dev libglfw3 libgles2-mesa-dev
```

Arena Setup

Create a dir `agent-arena-environment` and go into dir.

Follow the instructions for cloning and building Softgym.[1]

Follow the instructions for Agent-Arena.[2]

[1] https://github.com/halid1020/softgym/tree/master
[2] https://github.com/halid1020/agent-arena-v0

Docker Setup

I provide a Dockerfile based on Ubuntu 22.04 for machines running different versions of Ubuntu.
Build the image which is provided in the code:

```
docker build --network=host -t agent-arena-env .
```

Create and start a container with X11 GUI forwarding (all GPUs, host networking, volumes for code/data, auto-restart):

```
# Allow local X11 connections
xhost +local:root

docker run -d --gpus all --network=host \
  --name agent-arena-ml \
  --runtime=nvidia \
  --restart unless-stopped \
  --privileged \
  -e NVIDIA_VISIBLE_DEVICES=all \
  -e NVIDIA_DRIVER_CAPABILITIES=all \
  -e DISPLAY=$DISPLAY \
  -e QT_X11_NO_MITSHM=1 \
  -v /tmp/.X11-unix:/tmp/.X11-unix:rw \
  -v "$HOME/.Xauthority:$HOME/.Xauthority:rw" \
  -e CONDA_HOME=$HOME/miniconda3 \
  -v $HOME/miniconda3:$HOME/miniconda3 \
  -v $HOME/agent-arena-env/agent-arena-v0:/workspace/agent-arena-v0 \
  -v $HOME/agent-arena-env/softgym:/workspace/softgym \
  -v $HOME/agent-arena-env/ml_logs:/workspace/ml_logs \
  -v $HOME/agent-arena-env/checkpoints:/workspace/checkpoints \
  -v $HOME/agent-arena-env/data:/workspace/data \
  -w /workspace \
  agent-arena-env tail -f /dev/null
```

Notes:

- Run `xhost +local:root` *on the host* before launching the container to allow X11 connections.
- Adjust `CONDA_HOME` to your host path if needed (e.g., `$HOME/anaconda3`).
- Enter the container with: `docker exec -it agent-arena-ml bash`.
- GUI applications launched inside the container should appear on host display.
- If you get X11 authentication errors, ensure `$DISPLAY` and `.Xauthority` are correctly mounted and that `xhost` permissions are set.
- Expected host directory layout under `$HOME/agent-arena-env/`: `agent-arena-v0`, `softgym`, `ml_logs`, `checkpoints`, `data`.
- For SoftGym/Agent-Arena setup inside the container, follow their upstream README steps in `/workspace/softgym` and `/workspace/agent-arena-v0`.

Bibliography

[1] Claudio Urrea and John Kern, 'Recent advances and challenges in industrial robotics: A systematic review of technological trends and emerging applications,' *Processes*, 2025, DOI: 10.3390/pr13030832.

[2] Alberta Longhini, Yufei Wang, Irene Garcia-Camacho *et al.*, 'Unfolding the literature: A review of robotic cloth manipulation,' *Annual Review of Control, Robotics, and Autonomous Systems*, 5th May 2025, DOI: 10.1146/annurev-control-022723-033252.

[3] Halid Abdulrahim Kadi and Kasim Terzić, 'Data-driven robotic manipulation of cloth-like deformable objects: The present, challenges and future prospects,' *Sensors*, Jan. 2023, DOI: 10.3390/s23052389.

[4] Thomas M. Moerland, Joost Broekens, Aske Plaat *et al.*, *Model-based Reinforcement Learning: A Survey*. Now Publishers, 2023, ISBN: 978-1-63828-056-9, DOI: 10.1561/9781638280576.

[5] Danijar Hafner, Timothy Lillicrap, Ian Fischer *et al.*, 'Learning latent dynamics for planning from pixels,' in *Proceedings of International Conference on Machine Learning (ICML)*, 24th May 2019, [Online]. Available: https://proceedings.mlr.press/v97/hafner19a.html.

[6] Danijar Hafner, Timothy Lillicrap, Jimmy Ba *et al.*, 'Dream to control: Learning behaviors by latent imagination,' in *International Conference on Learning Representations (ICLR)*, 17th Mar. 2020, [Online]. Available: https://openreview.net/forum?id=S1lOTC4tDS.

[7] Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi *et al.*, 'Mastering atari with discrete world models,' in *International Conference on Learning Representations (ICLR)*, 2021, [Online]. Available: https://openreview.net/forum?id=0oabwyZbOu.

[8] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba *et al.*, 'Mastering diverse control tasks through world models,' *Nature*, 17th Apr. 2025, DOI: 10.1038/s41586-025-08744-2.

[9] Halid Abdulrahim Kadi and Kasim Terzić, 'PlaNet-ClothPick: Effective fabric flattening based on latent dynamic planning,' in *IEEE/SICE International Symposium on System Integration (SII)*, Jan. 2024, DOI: 10.1109/SII58957.2024.10417545.

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar *et al.*, 'Attention is all you need,' in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 4th Dec. 2017, ISBN: 978-1-5108-6096-4, [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[11] Chang Chen, Yi-Fu Wu, Jaesik Yoon *et al.*, *Transdreamer: Reinforcement learning with transformer world models*, 2022, arXiv: 2202.09481 [cs.LG].

[12] Weipu Zhang, Gang Wang, Jian Sun *et al.*, 'STORM: Efficient stochastic transformer based world models for reinforcement learning,' in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023, [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/5647763d4245b23e6a1cb0a8947b38c9-Paper-Conference.pdf.

[13] Maxime Burchi and Radu Timofte, 'Learning transformer-based world models with contrastive predictive coding,' in *International Conference on Learning Representations (ICLR)*, 4th Oct. 2024, [Online]. Available: https://openreview.net/forum?id=YK9G4Htdew.

[14] Vincent Micheli, Eloi Alonso and François Fleuret, 'Transformers are sample-efficient world models,' in *International Conference on Learning Representations (ICLR)*, 29th Sep. 2022, [Online]. Available: https://openreview.net/forum?id=vhFu1Acb0xb.

[15] Jan Robine, Marc Höftmann, Tobias Uelwer *et al.*, 'Transformer-based world models are happy with 100k interactions,' in *International Conference on Learning Representations (ICLR)*, 29th Sep. 2022, [Online]. Available: https://openreview.net/forum?id=TdBaDGCpjly.

[16] Halid Abdulrahim Kadi and Kasim Terzić, *Agent-Arena: A general framework for evaluating control algorithms*, 8th Apr. 2025, arXiv: 2504.06468[cs].

[17] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert *et al.*, 'Mastering atari, go, chess and shogi by planning with a learned model,' *Nature*, Dec. 2020, DOI: 10.1038/s41586-020-03051-4.

[18] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos *et al.*, 'Model-based reinforcement learning for atari,' in *International Conference on Learning Representations (ICLR)*, 2020, [Online]. Available: https://openreview.net/pdf?id=S1xCPJHtDB.

[19] David Ha and Jürgen Schmidhuber, 'World models,' 28th Mar. 2018, arXiv: 1803.10122.

[20] Fei Deng, Junyeong Park and Sungjin Ahn, 'Facing off world model backbones: RNNs, Transformers, and S4,' in *Advances in Neural Information Processing Systems (NeurIPS)*, ser. NIPS '23, 2023, [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/e6c65eb9b56719c1aa45ff73874de317-Paper-Conference.pdf.

[21] Martin L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming* (Wiley Series in Probability and Statistics), 1st ed. Wiley, 15th Apr. 1994, ISBN: 978-0-471-61977-2, DOI: 10.1002/9780470316887.

[22] Simon J. D. Prince, 'Reinforcement learning,' in *Understanding Deep Learning*. Cambridge, Massachusetts and London, England: The MIT Press, 2024, ISBN: 978-0-262-04864-4.

[23] Richard S. Sutton and Andrew Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, Massachusetts and London, England: The MIT Press, 2020, 526 pp., ISBN: 978-0-262-03924-6.

[24] Steven L. Brunton and J. Nathan Kutz, 'Reinforcement learning,' in *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, 2nd ed. Cambridge University Press, 5th May 2022, ISBN: 978-1-009-09848-9, DOI: 10.1017/9781009089517.016.

[25] Aske Plaat, Walter Kosters and Mike Preuss, 'High-accuracy model-based reinforcement learning, a survey,' *Artificial Intelligence Review*, 1st Sep. 2023, DOI: 10.1007/s10462-022-10335-w.

[26] Richard S. Sutton and Andrew Barto, 'Planning and learning with tabular methods,' in *Reinforcement learning: an introduction*, 2nd ed. Cambridge, Massachusetts and London, England: The MIT Press, 2020, ISBN: 978-0-262-03924-6.

[27] Diederik P. Kingma and Max Welling, 'Auto-encoding variational bayes,' in *International Conference on Learning Representations (ICLR)*, 2014, [Online]. Available: https://openreview.net/forum?id=33X9fd2-9FyZd.

[28] David Ha and Jürgen Schmidhuber, 'Recurrent world models facilitate policy evolution,' in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018, [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/2de5d16682c3c35007e4e92982f1a2ba-Abstract.html.

[29] Ibomoiye Domor Mienye, Theo G. Swart and George Obaido, 'Recurrent neural networks: A comprehensive review of architectures, variants, and applications,' *Information*, Sep. 2024, DOI: 10.3390/info15090517.

[30] Richard S. Sutton, 'Dyna, an integrated architecture for learning, planning, and reacting,' *SIGART Bull.*, 1st Jul. 1991, DOI: 10.1145/122344.122377.

[31] Eduardo F. Camacho, Carlos Bordons and José M. Maestre, *Model Predictive Control* (Advanced Textbooks in Control and Signal Processing), 3rd ed. Springer Cham, 2025, 359 pp., ISBN: 978-3-031-87596-0, [Online]. Available: https://link.springer.com/book/9783031875953.

[32] Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis *et al.*, 'Planning to explore via self-supervised world models,' in *Proceedings of International Conference on Machine Learning (ICML)*, 13th Jul. 2020, [Online]. Available: http://proceedings.mlr.press/v119/sekar20a/sekar20a.pdf.

[33] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel *et al.*, 'Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model,' in *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 6th Dec. 2020, ISBN: 978-1-7138-2954-6, [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/08058bf500242562c0d031ff830ad094-Abstract.html.

[34] Zihang Dai, Zhilin Yang, Yiming Yang *et al.*, 'Transformer-XL: Attentive language models beyond a fixed-length context,' in *Proceedings of Annual Meeting of the Association for Computational Linguistics (ACL)*, Jul. 2019, DOI: 10.18653/v1/P19-1285.

[35] Marc G. Bellemare, Yavar Naddaf, Joel Veness *et al.*, 'The arcade learning environment: An evaluation platform for general agents,' *Journal of Artificial Intelligence Research*, 14th Jun. 2013, DOI: 10.1613/jair.3912.

[36] Xingyu Lin, Yufei Wang, Jake Olkin *et al.*, 'SoftGym: Benchmarking deep reinforcement learning for deformable object manipulation,' in *Proceedings of Conference on Robot Learning (CoRL)*, 4th Oct. 2021, [Online]. Available: https://proceedings.mlr.press/v155/lin21a.html.

[37] Xiao Ma, David Hsu and Wee Sun Lee, 'Learning latent graph dynamics for visual manipulation of deformable objects,' in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2022, DOI: 10.1109/ICRA46639.2022.9811597.

[38] Andreas Doumanoglou, Jan Stria, Georgia Peleka *et al.*, 'Folding clothes autonomously: A complete pipeline,' *IEEE Transactions on Robotics*, Dec. 2016, DOI: 10.1109/TRO.2016.2602376.

[39] Kai Mo, Chongkun Xia, Xueqian Wang *et al.*, 'Foldsformer: Learning sequential multi-step cloth manipulation with space-time attention,' *IEEE Robotics and Automation Letters*, Feb. 2023, DOI: 10.1109/LRA.2022.3229573.

[40] Wilson Yan, Ashwin Vangipuram, Pieter Abbeel *et al.*, 'Learning predictive representations for deformable objects using contrastive estimation,' in *Proceedings of Conference on Robot Learning (CoRL)*, 4th Oct. 2021, [Online]. Available: https://proceedings.mlr.press/v155/yan21a.html.

[41] Xingyu Lin, Yufei Wang, Zixuan Huang *et al.*, 'Learning visible connectivity dynamics for cloth smoothing,' in *Conference on Robot Learning (CoRL)*, 19th Jun. 2021, [Online]. Available: https://openreview.net/forum?id=n1hDe9iK6ms.

[42] Alberta Longhini, Michael C. Welle, Zackory Erickson *et al.*, 'AdaFold: Adapting folding trajectories of cloths via feedback-loop manipulation,' *IEEE Robotics and Automation Letters*, Nov. 2024, DOI: 10.1109/LRA.2024.3436329.

[43] Ryan Hoque, Daniel Seita, Ashwin Balakrishna *et al.*, 'VisuoSpatial foresight for physical sequential fabric manipulation,' *Autonomous Robots*, 1st Jan. 2022, DOI: 10.1007/s10514-021-10001-0.

[44] Kyunghyun Cho, Bart Van Merrienboer, Caglar Gulcehre *et al.*, 'Learning phrase representations using RNN encoder–decoder for statistical machine translation,' in *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, DOI: 10.3115/v1/D14-1179.

[45] Steven L. Brunton and J. Nathan Kutz, 'Neural networks and deep learning,' in *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, 2nd ed. Cambridge University Press, 5th May 2022, ISBN: 978-1-009-09848-9, DOI: 10.1017/9781009089517.

[46] Sepp Hochreiter and Jürgen Schmidhuber, 'Long short-term memory,' *Neural Computation*, Nov. 1997, DOI: 10.1162/neco.1997.9.8.1735.

[47] Simon J. D. Prince, 'Transformers,' in *Understanding Deep Learning*. Cambridge, Massachusetts and London, England: The MIT Press, 2024, ISBN: 978-0-262-04864-4.

[48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky *et al.*, 'Dropout: A simple way to prevent neural networks from overfitting,' *Journal of Machine Learning Research (JMLR)*, 1st Jan. 2014.

[49] Djork-Arné Clevert, Thomas Unterthiner and Sepp Hochreiter, *Fast and accurate deep network learning by exponential linear units (ELUs)*, 22nd Feb. 2016, arXiv: 1511.07289[cs].

[50] Ian Sommerville, 'Project management,' in *Software engineering*, 10th ed. Boston: Pearson Ltd, 2016, ISBN: 978-1-292-09613-1.

[51] Marko Ikonen, Elena Pirinen, Fabian Fagerholm *et al.*, 'On the impact of kanban on software project work: An empirical case study investigation,' in *IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*, Apr. 2011, ISBN: 978-1-61284-853-2, DOI: 10.1109/ICECCS.2011.37.

[52] Andy Zeng, Pete Florence, Jonathan Tompson *et al.*, 'Transporter networks: Rearranging the visual world for robotic manipulation,' in *Proceedings of Conference on Robot Learning (CoRL)*, 4th Oct. 2021, [Online]. Available: https://proceedings.mlr.press/v155/zeng21a.html.

[53] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron *et al.*, 'dm_control: Software and tasks for continuous control,' *Software Impacts*, 1st Nov. 2020, DOI: 10.1016/j.simpa.2020.100022.

[54] Emanuel Todorov, Tom Erez and Yuval Tassa, 'MuJoCo: A physics engine for model-based control,' in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2012, DOI: 10.1109/IROS.2012.6386109.

[55] 'Bullet real-time physics simulation — home of bullet and PyBullet: Physics simulation for games, visual effects, robotics and reinforcement learning.' (21st Mar. 2022), [Online]. Available: https://pybullet.org/ (visited on 08/06/2025).

[56] Miles Macklin, Matthias Müller, Nuttapong Chentanez *et al.*, 'Unified particle physics for real-time applications,' *ACM Transactions on Graphics*, 27th Jul. 2014, DOI: 10.1145/2601097.2601152.

[57] Zhou Wang, A.C. Bovik, H.R. Sheikh *et al.*, 'Image quality assessment: From error visibility to structural similarity,' *IEEE Transactions on Image Processing*, Apr. 2004, DOI: 10.1109/TIP.2003.819861.