UNIVERSITÀ DEGLI STUDI DI TRIESTE

Dipartimento di Matematica e Geoscienze

Laurea Magistrale in Data Science and Artificial Intelligence

---

TESI DI LAUREA

Learning to Plan in Latent Space with Joint-Embedding Predictive Architectures

Relatore:
Prof. Luca Bortolussi
Correlatore:
Prof. Alfredo Canziani

Laureando:
Gabriele Gavino Pintus

A. A. 2024/2025

A chi c'era,
a chi c'è,
a chi ci sarà

Grazie

Sintesi

L'apprendimento di modelli interni del mondo è un requisito fondamentale per agenti autonomi capaci di previsione, ragionamento e pianificazione su orizzonti temporali estesi. Gli approcci tradizionali al world modeling operano direttamente nell'input space, il quale è ad alta dimensionalità, ridondante e solo parzialmente rilevante ai fini decisionali. Le Joint-Embedding Predictive Architectures (JEPA) offrono un'alternativa: apprendono a prevedere stati futuri in un representation space appreso che cattura soltanto le componenti prevedibili dell'ambiente, evitando la necessità di una ricostruzione completa.

In questa tesi, esploriamo l'uso di un modello JEPA per la navigazione goal-conditioned nell'ambiente PointMaze. Utilizziamo un encoder di immagini per estrarre rappresentazioni strutturate da osservazioni RGB grezze e introduciamo un meccanismo basato su maschere che separa gli elementi statici dello sfondo dalle informazioni dinamiche dipendenti dall'agente. Il predittore modella l'evoluzione temporale esclusivamente nella componente dinamica, producendo previsioni più robuste nel representation space.

Addestriamo il modello tramite obiettivi auto-supervisionati ispirati ai moderni metodi joint-embedding, includendo la tecnica dello stop-gradient resa popolare da SimSiam e ulteriori vincoli di coerenza temporale. Attraverso diversi esperimenti, mostriamo che il representation space appreso codifica la posizione con alta fedeltà e che la decomposizione statico-dinamica proposta migliora significativamente la prevedibilità e le prestazioni di pianificazione.

Infine, integriamo il modello JEPA in una pipeline di controllo basata su model-predictive control che esegue l'ottimizzazione della traiettoria interamente nel representation space. Il pianificatore risultante riesce a navigare con successo in labirinti complessi senza informazioni esplicite sullo stato, senza features progettate a mano e senza alcun segnale di supervisione. Questi risultati evidenziano il potenziale dei modelli predittivi joint-embedding come modelli del mondo compatti, efficienti e di uso generale per il controllo e il processo decisionale autonomo.

Abstract

Learning internal models of the world is a fundamental requirement for autonomous agents capable of prediction, reasoning, and long-horizon planning. Traditional world-modeling approaches operate directly in observation space, which is high-dimensional, redundant, and only partially relevant for decision making. Joint-Embedding Predictive Architectures (JEPA) offer an alternative: they learn to predict future states in a learned representation space that captures only the predictable components of the environment, avoiding full reconstruction.

In this thesis, we investigate the use of a JEPA world model for goal-conditioned navigation in the PointMaze environment. We design a visual encoder that extracts structured representations from raw RGB observations and introduce a mask-based mechanism that separates static background elements from dynamic agent-dependent information. The dynamics predictor then models temporal evolution exclusively in the dynamic component, yielding more robust predictions in representation space.

We train the model using self-supervised objectives inspired by modern joint-embedding methods, including the stop-gradient mechanism popularized by SimSiam and additional temporal consistency constraints. Through several experiments, we show that the learned representation space encodes position information with high fidelity and that the proposed static-dynamic decomposition substantially improves predictability and downstream planning performance, achieving 100% success rate with frequent replanning.

Finally, we integrate the JEPA model into a model-predictive control pipeline that performs trajectory optimization entirely in representation space. The resulting planner successfully navigates complex mazes without explicit state labels, handcrafted features, or task-specific supervision. These results highlight the potential of joint-embedding predictive models as compact, efficient, and general-purpose world models for control and autonomous decision making.

Contents

Chapter 1

Introduction

1.1 The Challenge of Learning World Models

The ability to predict the consequences of one's actions before executing them is a key feature of intelligent behavior. Humans and animals regularly anticipate how the world will evolve in response to their decisions, using these predictions to plan, avoid danger, and achieve goals efficiently. This capacity for mental simulation is built on what cognitive scientists and artificial intelligence researchers call a world model: an internal representation of the environment's structure and dynamics that allows an agent to reason about future states without direct interaction.

World models play a central role in theories of autonomous intelligence. An agent with an accurate model of its environment can simulate the outcomes of candidate actions, select effective plans, and adapt to new situations by relying on the underlying structure of the world. These abilities are crucial in complex or hazardous settings, where learning purely from trial and error would be too slow or unsafe.

Despite decades of research, giving artificial agents reliable world models remains a major challenge. Classical approaches often depend on hand-designed dynamics, which require domain expertise and do not scale to environments where the relevant state variables are unknown. Data-driven methods learn predictive models directly from experience, but struggle when observations are high-dimensional, redundant, or contain irrelevant and unpredictable information. Addressing these difficulties requires methods that learn compact, structured representations of observations while focusing only on the components of the environment that are predictable and useful for planning.

1.2 Prediction in Representation Space

These difficulties suggest a different approach. Instead of predicting directly in observation space, we can first learn a compressed representation that captures only the predictable and task-relevant aspects of the environment. The model then operates in

this abstract space, where the dynamics are simpler and irrelevant details have been removed.

This idea has appeared in many forms throughout machine learning, including state-space models, latent dynamics models, and learned simulators. What recent work highlights is that both the representation and its dynamics can be learned jointly, without explicit supervision about which features matter. Through training, the model discovers which aspects of the input should be encoded and which can be ignored.

Joint-Embedding Predictive Architectures (JEPA) are a promising example of this principle. Introduced by Yann LeCun as a path toward more flexible and human-like machine intelligence [14], JEPA models predict future states in a learned embedding space rather than reconstructing raw observations. By operating in representation space, they naturally focus on what is predictable and useful for planning while discarding noise and irrelevant variation.

This perspective is consistent with biological vision. The visual cortex does not store a pixel-level copy of the retinal image; it builds increasingly abstract features that support recognition and action. JEPA models follow a similar strategy, transforming raw sensory input into structured representations where prediction becomes more tractable.

1.3 Contributions of This Thesis

In this thesis, we explore the application of Joint-Embedding Predictive Architectures to learning world models for goal-conditioned navigation. Our testbed is the PointMaze environment, a domain where a point-mass agent must navigate through procedurally generated mazes using only visual observations and velocity measurements. The agent receives no explicit position information, no explicit map of the maze, and no reward signal during training. Everything must be inferred from the structure of the data itself.

The contributions of this thesis can be summarized as follows:

A mask-based mechanism for static-dynamic decomposition We introduce a learned mask that separates the visual representation into static components (walls, floor, background) and dynamic components (the agent, its immediate surroundings). This decomposition allows the dynamics predictor to focus exclusively on the parts of the scene that actually change over time, simplifying the prediction. The mask is learned end-to-end without any explicit supervision, emerging naturally from the training objectives.

A training framework combining multiple self-supervised objectives. We develop a training procedure that combines prediction loss, inverse dynamics regularization, temporal invariance constraints, and mask regularization. Following recent

work in self-supervised learning, we employ stop-gradient operations to prevent representation collapse. The resulting model learns meaningful representations that encode position information with high fidelity, even though position labels are never provided during training.

Integration with model-predictive control We demonstrate that the learned world model can be used for planning directly in representation space. By extracting approximate positions from the learned mask, we define a cost function for goal-reaching and optimize trajectories using Model Predictive Path Integral control. The resulting system successfully navigates complex mazes that were never seen during training, achieving near-perfect success rates with periodic replanning.

1.4 Outline of the Thesis

The thesis is organized as follows.

Chapter 2: Representation Learning. We begin by reviewing the theoretical foundations of representation learning, focusing on the information bottleneck principle and the multi-view setting that underlies modern self-supervised methods. We survey three main families of approaches: contrastive methods, regularized methods and latent bootstrapping methods. These techniques provide the building blocks for training JEPA models without collapse.

Chapter 3: World Models. This chapter introduces the control and reinforcement learning concepts that motivate our work. We describe Model Predictive Control and its optimization methods, both gradient-based and sampling-based. We then turn to reinforcement learning, covering the Markov Decision Process formalism, value functions, goal-conditioned policies, and offline learning. These frameworks provide the context for how learned world models can be used for planning and decision-making.

Chapter 4: Joint Embedding Predictive Architecture. We present the JEPA framework in detail, describing the encoder-predictor architecture and the training objectives that prevent representation collapse. We discuss both regularization-based approaches and teacher-student architectures. We also briefly introduce Hierarchical JEPA as a direction for capturing multi-scale temporal structure.

Chapter 5: Implementation and Experiments. This is the core technical chapter, where we describe our specific implementation for the PointMaze environment. We detail the neural network architectures, the mask extraction mechanism, the training objectives and their coefficients and the planning procedure. We present experimental results demonstrating successful static-dynamic disentanglement, accurate position estimation, and reliable goal-reaching across diverse maze configurations.

Chapter 6: Conclusions. We conclude by summarizing our findings, discussing the limitations of our approach, and outlining directions for future work. We reflect on what these results suggest about the potential of joint-embedding predictive models as general-purpose world models for autonomous agents.

Chapter 2

Representation Learning

Representation learning is a subfield of machine learning that focuses on automatically discovering useful features or representations from input data. These representations can then be used for various downstream tasks. For example, in computer vision, raw pixel values of an image can be transformed into higher-level features that capture important aspects of the image, such as edges, textures, or object parts. These features can then be used for tasks such as image classification, object detection, or semantic segmentation.

We can identify three main approaches to representation learning:

- **Supervised Learning:** train a model using labeled data to learn representations that are useful for a specific task, and then use the learned representations for other tasks.
- **Unsupervised Learning:** learn representations from unlabeled data by finding patterns or structures without (almost) any prior information.
- **Self-Supervised Learning:** learn representations by creating auxiliary tasks that do not require labeled data.

Among these approaches, Self-Supervised Learning (SSL) has gained significant attention in recent years due to its ability to train models with virtually unlimited amounts of unlabeled data, which can be easily obtained from the web. While SSL shares this advantage with unsupervised methods, it additionally allows the introduction of inductive biases that guide the learning process toward more meaningful and generalizable representations. This capability has led to remarkable success across a wide range of domains, including natural language processing (NLP) and computer vision (CV). In this chapter we introduce some of the fundamental concepts behind representation learning.

2.1 The information bottleneck principle

The information bottleneck principle provides an information-theoretic perspective for representation learning, by training an encoder network to retain all information that is relevant for predicting the label while minimizing the amount of other, excess information in the representation.

2.1.1 Supervised learning setting

The goal of representation learning is to find a distribution $p(\boldsymbol{z} \mid \boldsymbol{x})$ that maps an input observation $\boldsymbol{x} \in \mathcal{X}$ to a representation $\boldsymbol{z} \in \mathcal{Z}$, while capturing relevant characteristics of the data. In scenarios where the ultimate task is to predict a label $\boldsymbol{y} \in \mathcal{Y}$, we are interested in representations $\boldsymbol{z}$ that retain enough information to discriminate between different labels. If the representation $\boldsymbol{z}$ contains enough information about the input $\boldsymbol{x}$ to predict the label $\boldsymbol{y}$, we say that $\boldsymbol{z}$ is *sufficient* for $\boldsymbol{y}$.

Definition 1 (Sufficiency of a representation)**.** A representation $\boldsymbol{z}$ is sufficient for $\boldsymbol{y}$ if and only if

$$\mathbb{I}(\boldsymbol{x}; \boldsymbol{y} \mid \boldsymbol{z}) = 0 \iff \mathbb{I}(\boldsymbol{x}; \boldsymbol{y}) = \mathbb{I}(\boldsymbol{y}; \boldsymbol{z}), \tag{2.1}$$

The equivalence states that $\boldsymbol{z}$ is sufficient for predicting $\boldsymbol{y}$ when knowing $\boldsymbol{z}$ makes $\boldsymbol{x}$ and $\boldsymbol{y}$ conditionally independent, meaning that $\boldsymbol{z}$ preserves all the information about $\boldsymbol{y}$ that was originally present in $\boldsymbol{x}$.

Among all possible representations, we are interested in finding the one that retains the most relevant information for the task at hand while discarding any excess information that is not useful for the prediction of the label. The mutual information between the input $\boldsymbol{x}$ and the representation $\boldsymbol{z}$ can be decomposed in two complementary parts: superfluous information, which is not useful for predicting $\boldsymbol{y}$, and predictive information, which is relevant for predicting $\boldsymbol{y}$:

$$\mathbb{I}(\boldsymbol{x}; \boldsymbol{z}) = \underbrace{\mathbb{I}(\boldsymbol{x}; \boldsymbol{z} \mid \boldsymbol{y})}_{\text{superfluous information}} + \underbrace{\mathbb{I}(\boldsymbol{y}; \boldsymbol{z})}_{\text{predictive information}} . \tag{2.2}$$

We then define the optimal representation $\boldsymbol{z}^\star$ as the one that maximizes the predictive information while minimizing the superfluous information, leading to the following optimization problem:

$$\boldsymbol{z}^\star = \arg\max_{\boldsymbol{z}} \mathbb{I}(\boldsymbol{y}; \boldsymbol{z}) - \mathbb{I}(\boldsymbol{x}; \boldsymbol{z} \mid \boldsymbol{y}). \tag{2.3}$$

Assuming that the representation $\boldsymbol{z}$ is obtained from the input $\boldsymbol{x}$ through a function f parameterized by $\boldsymbol{\theta}$, $\boldsymbol{z} = f(\boldsymbol{x}; \boldsymbol{\theta})$, we can rewrite the optimization problem as:

$$\boldsymbol{\theta}^\star = \arg\max_{\boldsymbol{\theta}} \mathbb{I}(\boldsymbol{y}; f(\boldsymbol{x}; \boldsymbol{\theta})) - \mathbb{I}(\boldsymbol{x}; f(\boldsymbol{x}; \boldsymbol{\theta}) \mid \boldsymbol{y}). \tag{2.4}$$

However, we must remember that the *no free lunch theorem* always holds, meaning that there is no universally optimal representation that works for all possible tasks.

The optimization problem above can yield a useful representation only because we deliberately introduce an inductive bias that restricts the model's hypothesis space through the usage of the label $\boldsymbol{y}$. As a consequence, the representation $\boldsymbol{z}$ will be optimal for the specific task we are trying to solve but may not generalize well to others. This fundamental limitation is formally stated in the following theorem.

Theorem 1 (No free generalization)**.** Let $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ be random variables and let $\boldsymbol{z}'$ be a representation of $\boldsymbol{x}$ such that $\mathbb{I}(\boldsymbol{x}; \boldsymbol{z}) > \mathbb{I}(\boldsymbol{x}; \boldsymbol{z}')$. Then, it is always possible to find a label $\boldsymbol{y}$ for which the representation $\boldsymbol{z}'$ is not predictive of $\boldsymbol{y}$ while $\boldsymbol{z}$ is.

The theorem implies that only when *all* solutions to the optimization problem are equally optimal can we expect the learned representation $\boldsymbol{z}$ to generalize to any other possible task different from predicting $\boldsymbol{y}$. Otherwise, there will always exist a classification task for which the chosen representation $\boldsymbol{z}$ is not optimal.

2.1.2 Multi-view information bottleneck

Difficulties arise when we attempt to apply the information bottleneck principle to self-supervised learning. In this setting, we do not have access to any label $\boldsymbol{y}$, and thus we cannot directly optimize the objective function previously defined. This leads us to ask whether it is still possible to learn a useful representation $\boldsymbol{z}$ even in the absence of labels.

The key intuition is that, given an input domain $\mathcal{X}$ and a specified set of downstream tasks $\mathcal{D} = \{ d_k : \mathcal{X} \to \mathcal{Y}_k \}_{k=1}^{K}$, it may be possible to find a family of transformations $T = \{ t : \mathcal{X} \to \mathcal{X} \}$ that preserve the semantics relevant for all tasks in $\mathcal{D}$.

$$\forall d \in \mathcal{D}, \forall t \in T, \quad S(\boldsymbol{x} \mid d) = S(t(\boldsymbol{x}) \mid d) \quad \forall \boldsymbol{x} \in \mathcal{X}, \tag{2.5}$$

where $S(\cdot \mid d)$ denotes the semantic meaning with respect to task d and represents the optimal predictor. This enables us to discard superfluous variability while preserving the information essential to the tasks of interest.

As you may wonder, the challenge is imagining what the set of downstream tasks could be, and consequently, what invariances the representation should satisfy. For example, in images or text, literature has shown that masking parts of the input is an effective general-purpose transformation. This is because both images and text exhibit *local correlations* and *global structures* that can be captured by learning to predict missing parts from the surrounding context. The same transformation would be useless if the input features were independent from each other, as no information could be inferred from the visible parts to reconstruct the missing ones.

Example 1 (Image representation)**.** Consider as input data a set of images of cats and dogs. Suppose we are interested in the following downstream tasks:

- Classify the image as containing a cat or a dog.
- Detect if the animal is sitting or standing.

- Determine whether the animal is happy or sad.

Each task can be solved by a *specific* classifier C that takes the image as input and outputs the prediction $\tilde{\boldsymbol{y}}$ for the corresponding label $\boldsymbol{y}$. However, even if we do not have access to the labels, we can observe that these classifiers will share several invariances, such as small rotations of the input. Posture, breed or the environment in which the photo was taken are superfluous for the chosen downstream tasks, hence the representation $\boldsymbol{z}$ should be invariant to them.

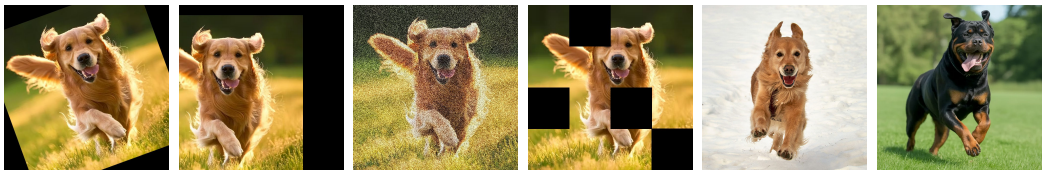


Figure 1: Examples of transformations that do not change the semantic meaning of the image: rotation, translation, noise, and masking, background change, and breed change.

We are looking for necessary conditions that, even if not sufficient, can help us restrict the hypothesis space and reduce the superfluous information in the representation $\boldsymbol{z}$. In other words, we must find a transformation T for which we can say: Whatever the label $\boldsymbol{y}$ we want to predict from the input $\boldsymbol{x}$ is, if we apply the transformation T to $\boldsymbol{x}$, the label $\boldsymbol{y}$ remains unchanged.

Before defining the optimization problem, let us first introduce some terms and definitions.

Definition 2 (View)**.** We call $\boldsymbol{v}$ a *view* of the input $\boldsymbol{x}$ to denote a transformation of the input data that does not alter its semantic meaning. It can be interpreted as a different "point of view" on the same data. For instance, if we are standing in front of a cat, we observe an image $\boldsymbol{x}$, but if we move to the right side of the cat, we observe a different image $\boldsymbol{x}'$, which is another view of the same cat.

Definition 3 (Redundancy)**.** We say that a view $\boldsymbol{v}_1$ is *redundant* with respect to another view $\boldsymbol{v}_2$ for $\boldsymbol{y}$ if the information contained in $\boldsymbol{v}_1$ does not provide any additional predictive information when $\boldsymbol{v}_2$ is already observed:

$$\mathbb{I}(\boldsymbol{y}; \boldsymbol{v}_1 \mid \boldsymbol{v}_2) = 0. \tag{2.6}$$

Definition 4 (Mutual redundancy)**.** We say that two views $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are *mutually redundant* if observing one of them does not provide any additional predictive information when the other is already observed.

Whenever two views are mutually redundant, the following proposition holds.

Proposition 1. Let $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ be two mutually redundant views of the input $\boldsymbol{x}$ and let $\boldsymbol{z}_1$ be a representation of $\boldsymbol{v}_1$. If $\boldsymbol{z}_1$ is sufficient for $\boldsymbol{v}_2$, then it is as predictive for $\boldsymbol{y}$ as the joint observation of the two views $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$:

$$\mathbb{I}(\boldsymbol{v}_1; \boldsymbol{v}_2 \mid \boldsymbol{z}_1) = 0 \implies \mathbb{I}(\boldsymbol{y}; \boldsymbol{z}_1) = \mathbb{I}(\boldsymbol{y}; \boldsymbol{v}_1, \boldsymbol{v}_2). \tag{2.7}$$

In other words, whenever it is possible to assume mutual redundancy, any representation that contains all the information shared by both views (the redundant information) is as predictive as their joint observation.

As in equation 2.2, we can decompose the mutual information between superfluous and predictive information in the multi-view setting:

$$\mathbb{I}(\boldsymbol{v}_1; \boldsymbol{z}_1) = \underbrace{\mathbb{I}(\boldsymbol{v}_1; \boldsymbol{z}_1 \mid \boldsymbol{v}_2)}_{\text{superfluous information}} + \underbrace{\mathbb{I}(\boldsymbol{v}_2; \boldsymbol{z}_1)}_{\text{predictive information for } \boldsymbol{v}_2}. \tag{2.8}$$

2.2 Training the Encoder

In machine learning literature, a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Z}$ that maps an input $\boldsymbol{x} \in \mathcal{X}$ to a representation $\boldsymbol{z} \in \mathcal{Z}$ is called an *encoder*. The central question is how to find an encoder that produces meaningful representations of the input data. Neural networks are a natural choice for this task, as they are universal function approximators and can be efficiently trained using gradient-based optimization. However, a delicate step is the definition of an appropriate loss function that guides the learning process toward the desired representation.

Two broad architectures are commonly used:

- **Encoder-Decoder (Autoencoder)** architectures,
- **Encoder-Only (Joint Embedding)** architectures.

2.2.1 Encoder-Decoder Architecture

The first and most common approach found in the literature is the use of an *autoencoder*, which performs prediction in input space. The encoder network $\text{enc}_\theta(\cdot)$ maps an input $\boldsymbol{x}$ to a latent representation $\boldsymbol{z}$, and the decoder network $\text{dec}_\phi(\cdot)$ maps $\boldsymbol{z}$ back to input space to reconstruct $\boldsymbol{x}$. The per-sample reconstruction loss is defined as:

$$L(\theta, \phi; \boldsymbol{x}) \;=\; C(\boldsymbol{x}, \text{dec}_\phi(\text{enc}_\theta(\boldsymbol{x}))) \;=\; C(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \tag{2.9}$$

where C is a cost function (e.g., Euclidean distance for continuous data or cross-entropy for discrete data).

To introduce an *inductive bias* and encourage the model to learn more meaningful representations, one often passes to the encoder a *view* $\boldsymbol{v}$ of the input $\boldsymbol{x}$. This view is

obtained by applying a transformation T to the input, *i.e.*, $\boldsymbol{v} = T(\boldsymbol{x})$. In this case, the reconstruction loss becomes:

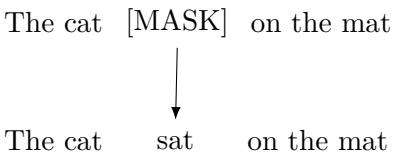
$$L(\theta, \phi; \boldsymbol{x}) \;=\; C(\boldsymbol{x}, \mathrm{dec}_\phi(\mathrm{enc}_\theta(T(\boldsymbol{x})))) \;=\; C(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \tag{2.10}$$

Since the model is forced to reconstruct the original input $\boldsymbol{x}$ from a potentially unbounded number of different views $\boldsymbol{v}$, the encoder must learn to be robust (ideally invariant) to the transformations applied by T, that is,

$$\mathrm{enc}_\theta(T(\boldsymbol{x})) \approx \mathrm{enc}_\theta(T'(\boldsymbol{x})), \quad \forall\, T, T' \sim \mathcal{T}. \tag{2.11}$$

Depending on the choice of transformation T, this approach has been referred to in the literature in different ways. For instance, if T is a masking transformation, the model has been called a *masked autoencoder*, if instead T adds noise to the input, like Gaussian noise, it has been called a *denoising autoencoder*. The general architecture remains the same, what changes is the type of transformation applied to the input.

A well-known example applied to text data is BERT [8]. Its main pretraining task, *masked language modeling* (MLM), consists of hiding a random subset of input tokens to the model and training it to predict the missing ones from the visible context.



The same masking principle has also been successfully applied to images, achieving state-of-the-art generalization performance at the time. By employing a high masking ratio, masked autoencoders train a Vision Transformer (ViT) to reconstruct the missing patches of pixels from the visible ones. The masking ratio regulates the tradeoff between global context and local details: higher masking ratios force the model to rely on more global structures while lower ratios allow it to focus on local patterns.

Figure 2: Examples of images reconstructed by a masked autoencoder from 25% of visible patches. From left to right: visible input, reconstructed output, original.

Depending on the semantic level of the input data, the architecture of the decoder and the type of masking can vary significantly. In the case of text data, each token

already represents a high-level semantic unit, so masking randomly dispersed tokens is sufficient to induce the model to learn meaningful representations, and a simple linear projector can be used as the decoder. Conversely, for image data, this approach would be less effective, as individual pixels do not carry significant semantic meaning, and the missing values can often be inferred by local averaging. Therefore, masking is applied to patches of pixels, which forces the model to learn more global structures and patterns in the data. A small ViT is typically used as the decoder to reconstruct the missing patches.

A common design choice is to make the decoder less expressive than the encoder since an excessively powerful decoder may lead the model to ignore the latent representation $\boldsymbol{z}$ and reconstruct the input directly from the visible parts, resulting in poor or trivial representations. This phenomenon, known as *posterior collapse*, is well documented in probabilistic autoencoders such as Variational AEs (VAEs).

Research indicates that encoder-decoder architectures yield low-level features, as their reconstruction objectives prioritize pixel-level details rather than high-level semantic structure, resulting in weaker and less robust representations [1, 18]. As a consequence, encoder-decoder methods often require additional supervised fine-tuning to achieve competitive performance on downstream tasks and are prone to overfitting in low-data regimes [1]. Furthermore, the need to reconstruct the input in its entirety imposes modality-specific loss designs, complicating the optimization process and limiting the generality of these approaches [3, 11]. In contrast, encoder-only methods formulate self-supervised learning as a *discriminative* problem in representation space, focusing on semantic alignment or latent prediction rather than generative reconstruction [1]. This shift avoids pixel-level biases and leads to more efficient and scalable learning. Finally, encoder-only methods have been shown to be more computationally efficient, typically requiring less memory and training time to achieve comparable or superior performance [18, 1].

2.2.2 Encoder-Only Architecture

An alternative to encoder-decoder approaches is offered by *encoder-only* architectures, which aim to train an encoder by predicting directly in the *representation space* rather than reconstructing the input. This family of models is commonly referred to in the literature as *Joint Embedding Architectures* (JEAs), as they jointly embed multiple augmented views of the same input into a shared representation space. The training objective acts directly on these embeddings, rather than in the input space, to encourage invariance to the applied transformations.

This approach generalizes naturally across different modalities, such as images, audio, and text, since it focuses on the alignment of high-level representations instead of pixel or token level reconstruction. This enables the direct extension of the framework to multi-modal settings, where different types of data is mapped by separate encoders into a shared representation space, and alignment is enforced through a suitable loss function.

While encoder-decoder architectures impose an inductive bias through reconstruction in input space, encoder-only approaches impose it directly at the representation level. However, training an encoder without an explicit reconstruction target introduces a new challenge: the learning objective must define a meaningful structure in the representation space. If the loss function merely enforces similarity between all views of the data, the encoder may converge to a *collapsed solution* in which all inputs are mapped to the same constant representation $\boldsymbol{z}$. In this degenerate case, the model trivially minimizes the loss but discards all information about the input, becoming invariant to any transformation.

Encoder-only methods (also referred to as joint-embedding methods) need additional mechanisms that preserve meaningful variability across samples while enforcing invariance across redundant views. Three main families of approaches can be identified:

- **Contrastive methods:** introduce negative samples to explicitly separate the representations of different inputs.
- **Regularized methods:** avoid collapse by imposing statistical constraints on the representation space (e.g., enforcing feature variance and decorrelation) instead of using negative samples.
- **Latent Bootstrapping methods:** leverage architectural or temporal asymmetries to prevent collapse, often using a momentum encoder or stop-gradient operations.

Before discussing these families, we introduce the general encoder-only SSL setup: the *Siamese architecture*. It consists of two encoders, $\text{enc}_\theta(\cdot)$ and $\text{enc}_\phi(\cdot)$, processing two augmented views $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ of the same input:

$$\boldsymbol{z}_1 = \text{enc}_\theta(\boldsymbol{v}_1), \qquad \boldsymbol{z}_2 = \text{enc}_\phi(\boldsymbol{v}_2). \tag{2.12}$$

The two encoders may share parameters ($\theta = \phi$) or remain separate, but in both cases they are trained to implement the same underlying function once learning converges. A loss function $C(\boldsymbol{z}_1, \boldsymbol{z}_2)$ directly compares the two embeddings, usually via a distance or similarity metric such as the Euclidean norm or cosine similarity, with the ideal outcome $\boldsymbol{z}_1 = \boldsymbol{z}_2$, reflecting invariance to the transformations used to generate $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$.

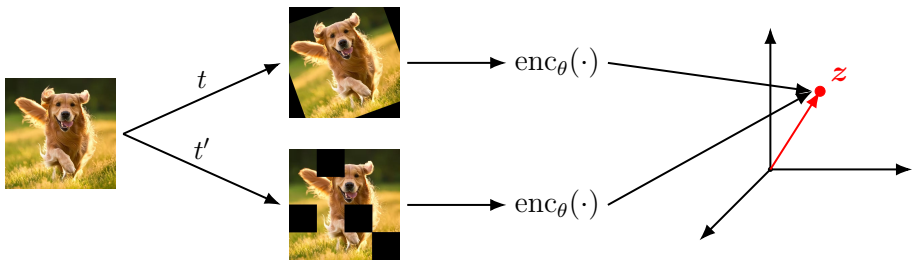


Figure 3: Siamese architecture with two encoders processing different views of the same input and producing the same latent representation.

Despite its simplicity, this setup makes the risk of collapse explicit: if nothing constrains the representation space beyond similarity between views, the model can minimize $C(\boldsymbol{z}_1, \boldsymbol{z}_2)$ by outputting the same constant representation for every input. The various encoder-only SSL methods differ precisely in how they introduce the necessary structure to avoid this trivial solution while still enforcing meaningful invariance.

Multimodal setting This framework is perfectly coherent with the multimodal setting, in which one can let the input space being the *space of concepts* and the transformations being mappings to different modalities. In this case the two encoders will seldom share weights as the architectures must be specialized for each modality.

2.2.2.1 Contrastive Learning

Contrastive learning prevents collapse by introducing *negative samples*: representations of different inputs that are explicitly pushed apart in representation space. As a consequence similar samples are pulled together while dissimilar ones are repelled, thereby shaping a discriminative embedding space as can be seen in figure 4.

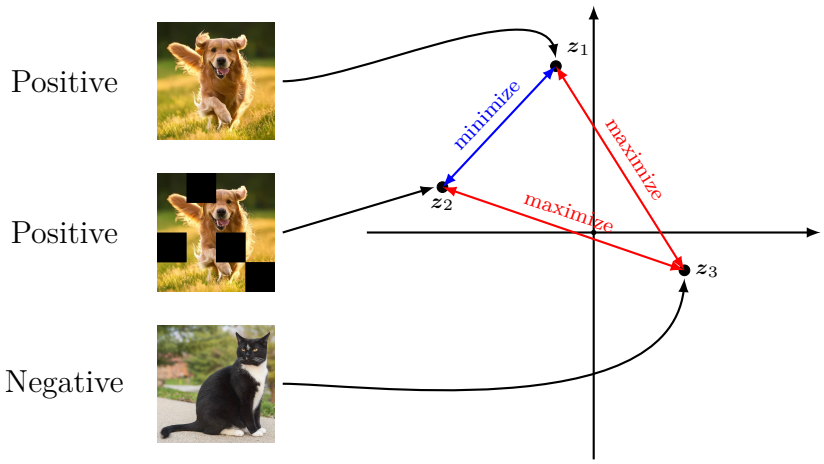


Figure 4: Contrastive learning encourages representations of similar samples to move closer together while pushing apart representations of dissimilar ones. The first two images show two augmented views of a dog (a positive pair), whereas the third image shows a cat (a negative sample). The objective pulls the dog representations together while repelling them from the cat representation, thereby shaping a discriminative embedding space.

An example of a contrastive objective that has been shown to work effectively is the *InfoNCE* loss, which originates from the *Noise-Contrastive Estimation (NCE)* principle proposed by Gutmann and Hyvärinen [10]. In NCE, a model learns to distinguish observed data from artificially generated noise through a logistic classification objective. This reframes the SSL setting to a supervised learning problem where the model is trained to discriminate between samples drawn from the data distribution and a

known noise distribution. This estimation principle yields consistent and asymptotically normal estimators even for unnormalized models and does not require explicit computation of the partition function, which is often intractable.

InfoNCE adapts this principle to the representation learning setting by defining *positive pairs*, two views of the same input, and *negative pairs*, views of different inputs, within a batch. The encoder is trained to correctly identify the positive pair among a set of negatives, which is equivalent to maximizing the log-likelihood of correctly classifying the positive sample in a contrastive discrimination task. Given a batch $\mathcal{B}$ of samples, the InfoNCE cost is expressed as:

$$C_{\text{InfoNCE}}(\boldsymbol{z}_1, \boldsymbol{z}_2) = -\log \frac{\exp(\beta \operatorname{sim}(\boldsymbol{z}_1, \boldsymbol{z}_2))}{\exp(\beta \operatorname{sim}(\boldsymbol{z}_1, \boldsymbol{z}_2)) + \sum_{k \in \mathcal{N}} \exp(\beta \operatorname{sim}(\boldsymbol{z}_1, \boldsymbol{z}_k))}, \tag{2.13}$$

where $\operatorname{sim}(\cdot, \cdot)$ denotes a similarity measure, such as the inner product, β is the inverse of the temperature and controls the sharpness of the similarity distribution, and $\mathcal{N}$ is the set of negative samples in the batch.

From an information-theoretic perspective, minimizing the InfoNCE cost can be interpreted as maximizing a lower bound on the mutual information between the two views of the same input. In practice, this objective has been shown to yield encoders that learn invariant and discriminative features, as demonstrated by methods such as SimCLR and MoCo. These approaches directly inherit the theoretical intuition of NCE: representation learning as a form of discrimination between informative samples (data) and uninformative ones (noise).

Despite their empirical success, contrastive methods face several challenges and limitations. Two of the most significant issues are discussed below.

Negative samples The effectiveness of contrastive learning strongly depends on the quality and quantity of negative samples. Since the goal is to push apart representations of distinct inputs, the method implicitly assumes that every other sample in the batch represents a truly different semantic entity. However, this assumption often fails in practice: two samples that share similar semantic content (e.g., two images of the same object class) may be mistakenly treated as negatives. This issue, known as *class collision* or the *false negative problem* [26, 20], can mislead the optimization process by pushing apart representations that should actually be close, thus degrading the learned feature space. Several works have proposed heuristics such as hard negative mining, adaptive sampling, or debiased contrastive objectives to mitigate this effect, but these solutions introduce additional complexity and hyperparameters.

Batch size Another major limitation of contrastive methods lies in their dependence on large batch sizes to provide the necessary number of negative examples to approximate the underlying data distribution. Empirically, models such as SimCLR [7] achieve optimal performance only with extremely large batches (often thousands of samples) or with auxiliary memory banks as in MoCo [12]. Small batches reduce the number of available negatives and lead to poor gradient estimates, slowing down convergence

and degrading representation quality. This requirement renders contrastive methods computationally expensive and memory-intensive, thereby hindering scalability and accessibility on limited hardware. Furthermore, large batch sizes are known to affect the optimization dynamics, often requiring careful tuning of the learning rate and coldness parameter β to maintain stability.

2.2.2.2 Regularized methods

Opposed to contrastive methods, regularized methods introduce explicit constraints on the representation space to maintain meaningful variability across different inputs. These constraints can be decomposed into two main parts: the first encourages *invariance* across redundant views, while the second promotes *diversity* among representations of different inputs.

Barlow Twins A notable example of this approach is the *Barlow Twins* method [27], which draws inspiration from the *redundancy reduction principle* in neuroscience proposed by H. Barlow [6]. He hypothesized that biological sensory systems reduce redundancy in their inputs by transforming correlated signals into statistically independent outputs, often called a factorial code, so that each neuron encodes unique, non-overlapping information. The method operates on two views, $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, of the same input, which are processed by two identical encoders to produce normalized embeddings $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathbb{R}^d$. Normalization can be achieved through an architectural choice such as Batch Normalization.

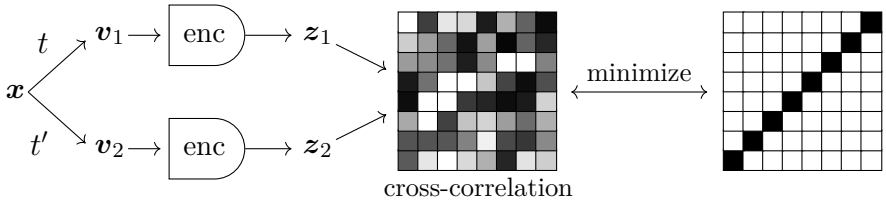


Figure 5: Schematic representation of the Barlow Twins architecture. Two augmented views of the same input, are encoded into normalized embeddings. The cross-correlation matrix between these embeddings is then computed and the objective function encourages its diagonal entries to be close to one (promoting invariance across views) while driving the off-diagonal entries toward zero (reducing redundancy across feature dimensions).

The training objective encourages the diagonal entries of the cross-correlation matrix $\boldsymbol{C}$ to approach one, ensuring that corresponding features in the two views are highly correlated, while penalizing off-diagonal terms to reduce redundancy across feature dimensions. Formally the cost is defined as:

$$C(z^{(1)}, z^{(2)}) = \underbrace{\sum_i (1 - \tilde{\Sigma}_{ii})^2}_{\text{invariance term}} + \underbrace{\lambda \sum_i \sum_{i \neq j} \tilde{\Sigma}_{ij}^2}_{\text{redundancy reduction term}} , \tag{2.14}$$

where λ is a hyperparameter that balances the two terms and $\tilde{\mathbf{\Sigma}} \in \mathbb{R}^{d \times d}$ is the estimated cross-correlation matrix across a batch of N samples:

$$\Sigma_{ij} = \frac{\sum_{b=1}^{N} z_{b,i}^{(1)} z_{b,j}^{(2)}}{\sqrt{\sum_{b=1}^{N} (z_{b,i}^{(1)})^2} \sqrt{\sum_{b=1}^{N} (z_{b,j}^{(2)})^2}}. \tag{2.15}$$

The first term of the loss enforces invariance to transformations, while the second acts as a decorrelation regularizer that maximizes the information content of the learned representation. The optimal solution of this objective corresponds to a correlation matrix equal to the identity, where each latent dimension is perfectly correlated with its counterpart across views and completely uncorrelated with all others. This combination ensures that features retain variability across dimensions while remaining stable across augmentations. Empirically, Barlow Twins achieves competitive performance with contrastive approaches such as SimCLR and BYOL, despite not relying on negative samples or momentum encoders.

VICReg A subsequent method, *Variance-Invariance-Covariance Regularization (VICReg)* [4], generalizes this principle by explicitly introducing three separate cost terms:

- **Invariance cost**, promoting similarity between two augmented views:

$$C_{\text{inv}}(\boldsymbol{z}, \boldsymbol{z}') = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \|\boldsymbol{z}_i - \boldsymbol{z}'_i\|^2. \tag{2.16}$$

- **Variance cost**, ensuring that each feature dimension maintains non-zero variance across the batch:

$$C_{\text{var}}(\boldsymbol{z}) = \frac{1}{d} \sum_{j=1}^{d} \max(0, \gamma - \text{std}(\boldsymbol{z}_j))^2. \tag{2.17}$$

- **Covariance cost**, decorrelating distinct feature dimensions:

$$C_{\text{cov}}(\boldsymbol{z}) = \frac{1}{d} \sum_{i \neq j} [\text{Cov}(\boldsymbol{z})]_{ij}^2 . \tag{2.18}$$

The total loss is then a weighted sum of these three components:

$$L_{\text{VICReg}} = C_{\text{inv}} + \mu C_{\text{var}} + \nu C_{\text{cov}}, \tag{2.19}$$

where μ and ν are hyperparameters that balance the contributions of each term, and γ is a target standard deviation. Since the standard deviation is computed across the batch, larger batches implicitly enforce a stronger diversity constraint: the model must maintain sufficient variance across a broader set of samples, effectively encouraging a more globally spread representation space. This behaviour acts as a soft repulsive effect

among embeddings, analogous to negative samples in contrastive learning, but achieved implicitly through batch-level statistics. Research is ongoing to better understand the precise role of batch size in these methods, its impact on the learned representations, and potential tuning strategies.

From an information-theoretic perspective, VICReg can be seen as a tractable approximation to maximizing the mutual information between representations of two views, Under the data distribution hypothesis, Schwartz et al. [21] showed that minimizing the VICReg loss yields the following lower bound to the mutual information between the two views' representations:

$$\mathbb{I}(\boldsymbol{z}, \boldsymbol{v}_2) \geq \mathbb{H}(\boldsymbol{z}) + \frac{d}{2}\log 2\pi - \frac{1}{2}\mathbb{E}_{\boldsymbol{v}_1,\boldsymbol{v}_2}\left[(\mu(\boldsymbol{v}_1) - \mu(\boldsymbol{v}_2))^2 + \log(|\Sigma(\boldsymbol{v}_1)| \cdot |\Sigma(\boldsymbol{v}_2)|)\right] \quad (2.20)$$

Empirical observations and limitations Regularized methods achieve performance on par with contrastive approaches while offering improved stability and lower computational demands, since they do not require large batches or memory banks. However, they rely heavily on batch-level statistics (variance and covariance estimates) and can become unstable with very small batches. Furthermore, the choice of loss coefficients (λ, μ, ν) and the dimensionality of the projection head can influence convergence and representation quality. Despite these limitations, redundancy-reduction methods represent a simple and principled path toward learning invariant, diverse, and information-rich representations without explicit contrastive mechanisms.

2.2.2.3 Latent Bootstrapping methods

The last family of encoder-only methods addresses the collapse problem not through negatives or explicit regularization, but through temporal or architectural asymmetry. These approaches rely on the principle of *latent bootstrapping*, where the network learns by predicting a slowly evolving or gradient-blocked version of itself, effectively creating self-consistent learning targets in representation space.

BYOL A prime example is Bootstrap Your Own Latent (BYOL), which showed that strong representations can be learned without negative samples or explicit regularization [9]. The method uses two networks sharing the same architecture: an *online student* and a *momentum-updated teacher*.

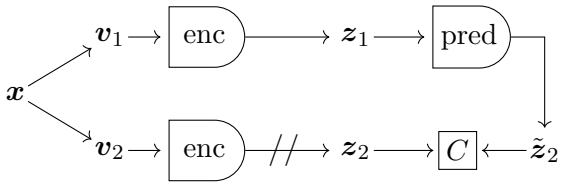


Figure 6: Schematic representation of the simplified BYOL architecture.

Given two augmented views $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ of the same input $\boldsymbol{x}$, the student and teacher encoders produce

$$\boldsymbol{z}_1 = \text{stud}_\theta(\boldsymbol{v}_1), \qquad \boldsymbol{z}_2 = \text{teach}_\phi(\boldsymbol{v}_2) \tag{2.21}$$

The teacher parameters ϕ are distilled from the student parameters θ through an exponential moving average (EMA):

$$\phi \leftarrow \tau\,\phi + (1-\tau)\,\theta, \tag{2.22}$$

where $\tau \in [0, 1)$ is a smoothing coefficient that controls the update speed. A small predictor is applied only on the student branch to predict the teacher output. The gradients are backpropagated only through the student network, while the teacher branch is stop-gradiented to prevent direct updates. Before computing the loss, both outputs are ℓ_2-normalized, turning the objective into a cosine-similarity regression problem. Let $s(\cdot, \cdot)$ denote cosine similarity, the BYOL cost is defined as

$$C(\boldsymbol{z}_1, \boldsymbol{z}_2) = -\tfrac{1}{2}\left[d(\text{pred}(\boldsymbol{z}_1), \text{sg}(\boldsymbol{z}_2)) + d(\text{pred}(\boldsymbol{z}_2), \text{sg}(\boldsymbol{z}_1))\right], \tag{2.23}$$

where $\text{sg}(\cdot)$ denotes the stop-gradient operator. The loss is symmetrized by swapping the two views and averaging.

Note In this simplified formulation, we remove the projector originally used in BYOL after the encoders. This is common in other variants, and collapse is still avoided thanks to the EMA teacher and the asymmetry introduced by the predictor.

SimSiam Following BYOL, SimSiam [7] further simplified the framework by removing the momentum teacher altogether. Instead, both branches share weights, and asymmetry is enforced purely through the *stop-gradient* operation on one side of the Siamese pair. The student branch learns to predict the stop-gradiented output of the other branch through a shallow predictor. Using the same cosine similarity $s(\cdot, \cdot)$, the SimSiam cost is defined as

$$C(\boldsymbol{z}_1, \boldsymbol{z}_2) = -\tfrac{1}{2}\left[s(\text{pred}(\boldsymbol{z}_1), \text{sg}(\boldsymbol{z}_2)) + s(\text{pred}(\boldsymbol{z}_2), \text{sg}(\boldsymbol{z}_1))\right]. \tag{2.24}$$

Without the stop-gradient operation, the model collapses to a constant representation, confirming that the asymmetry introduced by stopping the gradient is essential for stability. SimSiam achieves a top-1 accuracy of 68% on ImageNet using a ResNet-50 backbone.

The authors hypothesize that the non-collapsing behavior of SimSiam can be interpreted as an *Expectation-Maximization*-like (EM) process that alternates between optimizing the encoder parameters and the latent representations of each image. They consider the minimization of

$$L(\boldsymbol{\theta}, \boldsymbol{z}_{\boldsymbol{x}}) = \mathbb{E}_{\boldsymbol{x},t}\Big[\|\text{enc}_{\boldsymbol{\theta}}(t(\boldsymbol{x})) - \boldsymbol{z}_{\boldsymbol{x}}\|_2^2\Big], \tag{2.25}$$

where $\boldsymbol{z}_{\boldsymbol{x}}$ is a latent representation associated with image $\boldsymbol{x}$, and $\text{enc}_{\boldsymbol{\theta}}$ is the encoder. This objective can be solved by alternating between two subproblems: updating the encoder parameters $\boldsymbol{\theta}$ while keeping $\boldsymbol{z}$ fixed, and then updating $\boldsymbol{z}$ to be the expected

encoder output under random augmentations. The stop-gradient operation can thus be interpreted as enforcing this alternation implicitly, by preventing gradients from flowing into the latent targets during the encoder update. In practice, SimSiam performs only a *single* gradient descent update per alternation, which corresponds to the one-step approximation:

$$\boldsymbol{\theta}^{n+1} \leftarrow \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x},T}\Big[\|\mathrm{enc}_{\boldsymbol{\theta}}(t(\boldsymbol{x})) - \mathrm{enc}_{\boldsymbol{\theta}^n}(t'(\boldsymbol{x}))\|_2^2\Big], \tag{2.26}$$

with t and t' denoting two independent augmentations of the same image. Under this view, the predictor plays the role of approximating the expectation over augmentations that is otherwise ignored, effectively learning to map one view to the expected representation of another. Despite this intuition providing a plausible explanation for SimSiam's stability, no formal proof exists that this mechanism prevents collapse in all cases, nor that it simulates an EM process.

Chapter 3

World Models

The ability to predict the consequences of one's actions is a fundamental aspect of intelligence. In both humans and animals, such predictive capabilities arise from internal *world models*: structured representations of the environment that model its dynamics and patterns. These models allow an agent not only to anticipate future events but also to reason, plan, and act efficiently in previously unseen situations.

As highlighted by Yann LeCun in his position paper *A Path Towards Autonomous Machine Intelligence* [14], learning world models is a central component of autonomous systems. He argues that humans and animals acquire vast background knowledge through passive observation, constructing internal models that encode what is possible, plausible, or impossible. Such models enable rapid skill acquisition, few-shot generalization, and safe exploration, all abilities that current Artificial Intelligence systems lack.

Bridging this biological intuition with computational modeling motivates two complementary lines of work: *Model Predictive Control* (MPC), which plans using known dynamics and a designer-specified cost, and *Reinforcement Learning* (RL), which learns to act from experience when dynamics models or objectives are unknown or uncertain. In this chapter we introduce both paradigms, highlighting their connections and differences, and how they can be integrated with each other.

3.1 Model Predictive Control

In many control and decision-making problems, a model of the environment and a well-defined cost are available. This enables an agent to simulate the consequences of candidate actions and select those that minimize the cost. We consider a discrete-time dynamical system of the form

$$\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t), \qquad \boldsymbol{s}_t \in \mathcal{S},\ \boldsymbol{a}_t \in \mathcal{A}, \tag{3.1}$$

where

- $\boldsymbol{s}_t$ denotes the *state* at time t,
- $\boldsymbol{a}_t$ denotes the *action*,
- $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ defines the *system dynamics*,
- $\mathcal{S}$ and $\mathcal{A}$ are the *admissible state* and *action spaces*, respectively.

3.1.1 Cost Function Formulation

In most Model Predictive Control formulations, the *finite-horizon cost* is defined additively over the prediction horizon:

$$C(\boldsymbol{s}_t, \boldsymbol{a}_t, \ldots, \boldsymbol{s}_{t+N}) = \sum_{n=0}^{N-1} C_{\mathrm{s}}(\boldsymbol{s}_{t+n}, \boldsymbol{a}_{t+n}) + C_{\mathrm{f}}(\boldsymbol{s}_{t+N}), \tag{3.2}$$

where $C_{\mathrm{s}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *stage cost* and $C_{\mathrm{f}} : \mathcal{S} \to \mathbb{R}$ is the *final cost* or *terminal cost*. The final cost encourages convergence toward a reference or steady state often called goal or target, while the stage cost penalizes deviations from desired intermediate states or excessive control effort. Other variants of MPC may employ non-additive or stochastic cost functionals, such as

$$C = \mathbb{E}\left[\sum_{n=0}^{N-1} C_{\mathrm{s}}(\boldsymbol{s}_{t+n}, \boldsymbol{a}_{t+n})\right] \quad \text{or} \quad C = \max_{n<N} C_{\mathrm{f}}(\boldsymbol{s}_{t+n}, \boldsymbol{a}_{t+n}), \tag{3.3}$$

depending on whether the formulation is risk-neutral, risk-averse, or worst-case oriented. Finally, constraints on states and actions can be incorporated directly into the cost function. We may be interested in reaching a target state while considering only a subset of its components, or we may wish to penalize actions that exceed a certain threshold only along specific dimensions. In such cases, the cost function is designed to ignore irrelevant components and focus exclusively on those that matter. For instance, in a navigation task, the goal might be defined solely in terms of the spatial coordinates (x, y), while the velocity components (v_x, v_y) are disregarded.

3.1.2 Execution and Receding Horizon Policy

At each time step t, given the current state $\boldsymbol{s}_t$, MPC solves a finite-horizon trajectory optimization problem:

$$\begin{aligned} \min_{\boldsymbol{a}_{t:t+N-1}} \quad & C\Big(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}, \ldots, \boldsymbol{s}_{t+N}\Big) \\ \text{s.t.} \quad & \boldsymbol{s}_{t+n+1} = f(\boldsymbol{s}_{t+n}, \boldsymbol{a}_{t+n}), && n = 0, \ldots, N-1, \\ & \boldsymbol{s}_{t+n} \in \mathcal{S}, \quad \boldsymbol{a}_{t+n} \in \mathcal{A}, && n = 0, \ldots, N-1, \\ & \boldsymbol{s}_t \text{ given}, \end{aligned} \tag{3.4}$$

where N is the prediction horizon and $C : (\mathcal{S} \times \mathcal{A})^N \times \mathcal{S} \to \mathbb{R}$ is a cost function over the predicted trajectory. Since only the initial state and the goal states are specified, to predict future states given an action sequence $\boldsymbol{a}_{t:t+N-1}$, a model of the system dynamics f is required. If the exact dynamics are unknown, an approximate or learned model $\tilde{f}$

can be used instead. Obviously, the quality of the model directly affects the accuracy of the predictions and the resulting control performance.

Given:

- the initial state $\boldsymbol{s}_0$,
- the goal state $\boldsymbol{s}_g$,
- the dynamics model $\tilde{f}$,
- the cost function C,
- the prediction horizon N,
- an initial guess for the action sequence $\boldsymbol{a}^0_{0:N-1}$,

the optimization process *rolls out* the dynamics model over N steps to predict the state trajectory

$$\{\boldsymbol{s}_0, \tilde{\boldsymbol{s}}_1, \tilde{\boldsymbol{s}}_2, \ldots, \tilde{\boldsymbol{s}}_N\}, \tag{3.5}$$

then compute the cost function

$$C^n\Big(\boldsymbol{s}_0, \boldsymbol{a}^n_0, \tilde{\boldsymbol{s}}_1, \boldsymbol{a}^n_1, \ldots, \tilde{\boldsymbol{s}}_N\Big), \tag{3.6}$$

and finally update the action sequence $\boldsymbol{a}^n_{0:N-1}$ to reduce the cost

$$\boldsymbol{a}^{n+1}_{0:N-1} = \text{optimizer}\Big(\boldsymbol{a}^n_{0:N-1},\ C^n\Big(\boldsymbol{s}_0, \boldsymbol{a}^n_0, \tilde{\boldsymbol{s}}_1, \boldsymbol{a}^n_1, \ldots, \tilde{\boldsymbol{s}}_N\Big)\Big). \tag{3.7}$$

Let $\boldsymbol{a}^\star_{0:N-1}$ be the optimal sequence of actions found by the optimization process. Only the first K actions of this sequence are executed, after which the system state is updated to $\boldsymbol{s}_{t+K}$ hence using the true dynamics and the optimization process is restarted from the currently reached state. This defines a *K-step receding-horizon policy* with the special case $K = 1$ recovering the standard MPC formulation. The case in which $K = N$ corresponds to the open-loop execution of the entire optimized sequence, without any feedback. This practice of replanning helps reducing the exponentially growing prediction errors over long horizons. The choice of K balances computational cost and solution quality: shorter K implies more frequent replanning, which improves robustness but increases computational load.

3.1.3 Optimization Methods for MPC

Depending on whether the dynamics model $\tilde{f}$ and the cost function C are differentiable, different optimization strategies can be employed to solve the finite-horizon problem in (3.4). We distinguish between *gradient-based* and *gradient-free* approaches.

Gradient-based optimization When both $\tilde{f}(\boldsymbol{s}, \boldsymbol{a})$ and $C(\boldsymbol{s}, \boldsymbol{a})$ are differentiable, gradients of the cumulative cost with respect to the action sequence $\boldsymbol{a}_{t:t+N-1}$ can be computed efficiently by *backpropagation through time* (BPTT) on the unrolled dynamics. Depending on the complexity of the model, automatic differentiation tools or analytic expressions can be used. If the dynamics model is approximated by a neural

network, modern gradient-based optimizers as gradient descent or adaptive variants (e.g., Adam, RMSProp) update the action sequence according to

$$\boldsymbol{a} \leftarrow \boldsymbol{a} - \eta \nabla_{\boldsymbol{a}} C_t, \tag{3.8}$$

plus the additional momentum or adaptive terms, depending on the chosen algorithm. When curvature information is available, quasi-Newton or structure-exploiting methods such as *L-BFGS*, *iLQR*, or *Differential Dynamic Programming (DDP)* achieve faster convergence by locally linearizing the dynamics and quadratically approximating the cost.

Gradient-free optimization When the system dynamics or cost function are non-differentiable, *gradient-free* methods are employed. These approaches rely on sampling and evaluation instead of analytic gradients and can be described by the generic *Evolutionary Strategy (ES) cycle*:

1. **Sampling:** generate a population of candidate action sequences $\{\boldsymbol{a}_n\}_{n=1}^{N}$ from a search distribution $p_n(\boldsymbol{a})$.
2. **Evaluation:** execute the rollout under the dynamics model f to obtain the predicted state trajectory and compute the cost for each candidate solution.
3. **Selection:** identify the best-performing candidates according to their costs.
4. **Update:** adapt the distribution parameters to favor low-cost regions.

This process iteratively concentrates the search around promising regions of the action space. Modern stochastic optimizers such as the *Cross-Entropy Method* (CEM) and *Model Predictive Path Integral Control* (MPPI) are well-known strategies of this framework.

Cross-Entropy Method At iteration i, CEM samples a batch of N candidate action sequences $\{\boldsymbol{a}_n\}_{n=1}^{N} \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. It then selects the top-M *elite* samples with the lowest costs, forming the elite set $\mathcal{E}_i$. The distribution parameters are updated as

$$\boldsymbol{\mu}_{i+1} = \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{a}^{(m)}, \qquad \boldsymbol{\Sigma}_{i+1} = \frac{1}{M} \sum_{m=1}^{M} \Big(\boldsymbol{a}^{(m)} - \boldsymbol{\mu}_{i+1}\Big)\Big(\boldsymbol{a}^{(m)} - \boldsymbol{\mu}_{i+1}\Big)^{\top}, \tag{3.9}$$

where $\boldsymbol{a}^{(m)} \in \mathcal{E}_i$ denotes the m-th elite sample. To reduce computational cost, the covariance matrix $\boldsymbol{\Sigma}_i$ is often constrained to be diagonal or low-rank. Soft variants replace hard elite selection with Boltzmann-weighted averaging, assigning weights proportional to $\exp(-C(\boldsymbol{a})/\lambda)$, where λ controls the exploration-exploitation trade-off.

Model Predictive Path Integral Control *MPPI* perturbs an action sequence $\boldsymbol{a}_{t:t+N-1}$ with Gaussian noise $\epsilon_n \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$ to generate N sampled trajectories. The cost of the n-th trajectory is denoted $C_t(\boldsymbol{a}_n)$, and the update is obtained as a cost-weighted average of the perturbations:

$$\boldsymbol{a}^{\text{new}}_{t:t+N-1} = \boldsymbol{a}_{t:t+N-1} + \frac{\sum_{n=1}^{N} \exp\Big(-\beta\, C_t(\boldsymbol{a}_n)\Big)\, \epsilon_n}{\sum_{n=1}^{N} \exp\Big(-\beta\, C_t(\boldsymbol{a}_n)\Big)}. \tag{3.10}$$

Low-cost trajectories contribute more to the update, biasing the search toward favorable regions, while the coldness parameter β (also known as inverse temperature) regulates the balance between exploration and exploitation. For stochastic dynamics, the control objective is to minimize the expected cost over trajectories induced by the action sequence:

$$\min_{\boldsymbol{a}_{t:t+N-1}} \mathbb{E}_{x_{t+1:t+N} \sim p(\cdot|x_t, \boldsymbol{a}_{t:t+N-1})} \Big[C_t\Big(x_{t:t+N},\ \boldsymbol{a}_{t:t+N-1}\Big) \Big], \tag{3.11}$$

which can be estimated via Monte Carlo rollouts of the stochastic dynamics model.

3.1.4 The Truck Backer-Upper Problem

A classical example of planning optimization is the *Truck Backer-Upper* problem, originally introduced by Nguyen and Widrow [17]. It illustrates the challenges of controlling a nonholonomic, underactuated system whose dynamics are nonlinear and strongly coupled. The task is to back a trailer into a loading dock from arbitrary initial configurations by adjusting only the steering angle of the truck.

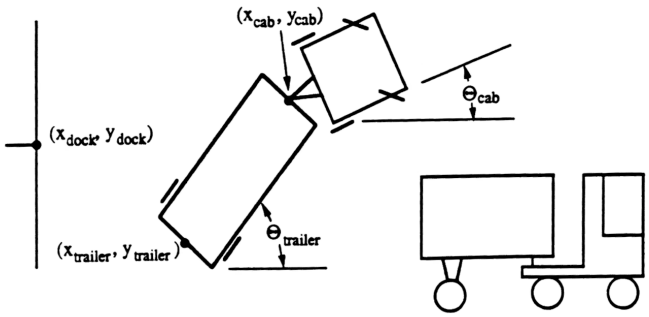


Figure 1: Schematic of the Truck Backer-Upper setup (truck, trailer, and loading dock).

Following the original formulation, the discrete-time index $n = 0, 1, \ldots$ denotes the control cycle. The system state is

$$\boldsymbol{s}_n = \Big(x_n,\ y_n,\ \theta_n^{\text{cab}},\ \theta_n^{\text{art}}\Big), \tag{3.12}$$

where (x_n, y_n) are the trailer axle centre coordinates, θ_n^{cab} is the truck orientation, and θ_n^{art} is the articulation angle between truck and trailer. The control input, hereafter referred to as the *action*, is the steering command

$$a_n \in [-1, 1], \tag{3.13}$$

with $a_n = -1$ and $a_n = +1$ corresponding to maximum left and right steering, respectively. The goal is to align the trailer with the dock at position $(x_{\text{dock}}, y_{\text{dock}})$, satisfying

$$x_{\text{trailer}} \approx x_{\text{dock}}, \qquad y_{\text{trailer}} \approx y_{\text{dock}}, \qquad \theta^{\text{trl}} \approx 0. \tag{3.14}$$

At each control cycle, the truck and trailer move a fixed backward distance, determined by the steering action. The discrete-time kinematic model is smooth and differentiable, which is ideal for gradient-based optimization.

In this setting, the objective is manually defined: we can engineer a cost function that reflects the task requirements, because the underlying goal—backing the trailer to the dock—is well understood and easily specified in geometric terms. A quadratic stage cost of the form

$$C_{\mathrm{s}}(\boldsymbol{s}_n, a_n) = (\boldsymbol{s}_n - \boldsymbol{s}_{\mathrm{goal}})^\top \boldsymbol{Q}\,(\boldsymbol{s}_n - \boldsymbol{s}_{\mathrm{goal}}) + R\,a_n^2, \tag{3.15}$$

is a common and convenient choice, together with a terminal cost

$$C_{\mathrm{f}}(\boldsymbol{s}_{n+N}) = (\boldsymbol{s}_{n+N} - \boldsymbol{s}_{\mathrm{goal}})^\top \boldsymbol{P}\,(\boldsymbol{s}_{n+N} - \boldsymbol{s}_{\mathrm{goal}}), \tag{3.16}$$

where

$$\boldsymbol{s}_{\mathrm{goal}} = \Big(x_{\mathrm{dock}},\, y_{\mathrm{dock}},\, \theta^{\mathrm{cab}}_{\mathrm{goal}},\, 0\Big), \tag{3.17}$$

and $\boldsymbol{Q} \succeq 0$ and $\boldsymbol{P} \succeq 0$ are weighting matrices and $R > 0$ is a scalar weight. Different cost formulations may yield distinct control behaviors: there is no unique cost function for this task. However, because the objective can be explicitly stated, minimizing positional error, ensuring alignment, and penalizing excessive steering, hand-crafting C_{s} and C_{f} is both feasible and effective in this context.

Because the dynamics are known exactly, the optimization problem can be solved as an open-loop trajectory optimization from beginning to end. Replanning would lead to no increase in performance and would only waste computational resources.

3.2 Reinforcement Learning

The framework of Model Predictive Control provides a general strategy to plan and act when both the system dynamics f are known, or easily approximated. However, in many scenarios the function f is only partially specified or entirely unknown. When an agent must learn how to behave solely from experience, without access to an explicit model or handcrafted objective, the problem naturally transitions to the domain of *Reinforcement Learning* (RL).

In this setting, the agent can no longer rely on a predefined cost function that explicitly tells it both where to go and how to get there. Instead, it receives a weaker training signal in the form of rewards, which only indicate whether an outcome is desirable or not. The agent must therefore learn an *internal surrogate of the objective* from interaction, formalized through the *value function*, which estimates how good it is to be in a particular state, or to take a particular action, given the long-term consequences that follow. Whereas MPC minimizes a known cumulative cost, reinforcement learning must infer and optimize the expected cumulative reward (or negative cost-to-go) through trial and error, gradually shaping a policy that achieves desirable outcomes without explicit supervision.

3.2.1 The Markov Decision Process

A fundamental mathematical framework for sequential decision-making is the *Markov Decision Process* (MDP), defined as the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ is the set of possible *states* of the environment,
- $\mathcal{A}$ is the set of *actions* available to the agent,
- $P(s' \mid s, a)$ is the *transition probability* describing the environment's dynamics,
- $R(s, a, s')$ is the *reward function* providing immediate feedback after taking action a in state s and reaching s',
- $\gamma \in [0, 1)$ is the *discount factor* weighing the importance of future rewards.

At each time step t, the agent observes the current state $\boldsymbol{s}_t$, selects an action $\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$ according to its policy π, transitions to the next state $\boldsymbol{s}_{t+1} \sim P(\cdot \mid \boldsymbol{s}_t, \boldsymbol{a}_t)$, and receives a reward $r_t = R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})$. The *Markov property* ensures that the future evolution of the process depends only on the current state and action:

$$p(\boldsymbol{s}_{t+1}, r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \ldots) = p(\boldsymbol{s}_{t+1}, r_t \mid \boldsymbol{s}_t, \boldsymbol{a}_t). \tag{3.18}$$

The policy $\pi(\boldsymbol{a} \mid \boldsymbol{s})$ defines a probability distribution over actions given a state. The agent's goal is to discover a policy that maximizes the expected discounted return:

$$J(\pi) = \mathbb{E}_{\pi, P}\left[\sum_{t=0}^{\infty} \gamma^t R(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1})\right], \tag{3.19}$$

where the expectation is taken over trajectories induced by π and the transition dynamics P.

Policy- and Value-based Approaches Reinforcement learning algorithms can pursue this goal in two principal ways. *Policy-based* methods approximate the policy $\pi_\theta(\boldsymbol{a} \mid \boldsymbol{s})$ directly and adjust its parameters to maximize $J(\pi_\theta)$ via gradient-based optimization. Alternatively, *value-based* methods approximate the value function or the action-value function, using them to derive an improved policy through greedy or soft-greedy selection. Modern strategies combine both approaches, learning a policy and a value function simultaneously to provide lower variance gradient estimates and more stable learning.

3.2.2 Learning the Objective

Since the agent receives only evaluative feedback in the form of rewards, it must *learn an internal surrogate of the objective* from experience. This surrogate is formalized through the *value function*, which estimates the expected cumulative reward obtainable from a given state or state-action pair under a policy π. Formally, the *state-value function*

$V^\pi(\boldsymbol{s})$ and the *action-value function* $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ are defined as:

$$V^\pi(\boldsymbol{s}) = \mathbb{E}_{\pi,P}\left[\sum_{k=0}^{\infty} \gamma^k R(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}) \,\middle|\, \boldsymbol{s}_t = \boldsymbol{s}\right], \tag{3.20}$$

$$Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\pi,P}\left[\sum_{k=0}^{\infty} \gamma^k R(\boldsymbol{s}_{t+k}, \boldsymbol{a}_{t+k}, \boldsymbol{s}_{t+k+1}) \,\middle|\, \boldsymbol{s}_t = \boldsymbol{s}, \boldsymbol{a}_t = \boldsymbol{a}\right]. \tag{3.21}$$

These functions are related by

$$V^\pi(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s})}[\, Q^\pi(\boldsymbol{s}, \boldsymbol{a})\,], \tag{3.22}$$

From a control-theoretic viewpoint, $-V^\pi(\boldsymbol{s})$ can be interpreted as a *cost-to-go* function: the expected future cost when following policy π from state $\boldsymbol{s}$. While classical control defines this cost explicitly, reinforcement learning estimates it from interaction through the reward signal.

The value function satisfies a recursive relationship known as the *Bellman equation*, which enforces temporal consistency across successive states:

$$V^\pi(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s}),\, \boldsymbol{s}'\sim P(\cdot|\boldsymbol{s},\boldsymbol{a})}[R(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V^\pi(\boldsymbol{s}')]\,, \tag{3.23}$$

$$Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\boldsymbol{s}'\sim P(\cdot|\boldsymbol{s},\boldsymbol{a}),\, \boldsymbol{a}'\sim\pi(\cdot|\boldsymbol{s}')}[R(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma Q^\pi(\boldsymbol{s}', \boldsymbol{a}')]\,. \tag{3.24}$$

These equations embody the principle of *dynamic programming*: the value of a state equals its immediate reward plus the discounted value of its successor.

Relation to Optimal Control The optimal value functions satisfy

$$V^\star(\boldsymbol{s}) = \max_{\boldsymbol{a}} Q^\star(\boldsymbol{s}, \boldsymbol{a}), \qquad Q^\star(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_{\boldsymbol{s}'\sim P(\cdot|\boldsymbol{s},\boldsymbol{a})}[R(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') + \gamma V^\star(\boldsymbol{s}')]\,, \tag{3.25}$$

with the corresponding optimal policy

$$\pi^\star(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} Q^\star(\boldsymbol{s}, \boldsymbol{a}). \tag{3.26}$$

These optimality equations parallel the optimization step in MPC, where actions are selected to minimize the predicted cumulative cost. In the continuous-time limit, they reduce to the Hamilton-Jacobi-Bellman (HJB) equations defining the optimal cost-to-go in continuous control systems.

3.2.3 Goal-Oriented Reinforcement Learning

In standard reinforcement learning, the reward function implicitly encodes the goal state the agent must reach. However, in many practical settings we would like our agent to be able to reach multiple different goals, specified at runtime. In such cases, *Goal-Oriented Reinforcement Learning* (also called *Goal-Conditioned* or *universal* RL) generalizes the MDP formalism by conditioning both the policy and the value function on a goal state variable $g \in \mathcal{G}$.

A goal-augmented MDP is defined by the tuple

$$\mathcal{M}_g = (\mathcal{S}, \mathcal{A}, \mathcal{G}, P, R_g, \gamma), \tag{3.27}$$

where $\mathcal{G}$ denotes the goal space and the reward function $R_g(s, a, s')$ depends on the current goal. The agent's objective is to learn a single policy that can reach many possible goals:

$$\pi_\theta(a \mid s, g) = \text{probability of taking action } a \text{ in state } s \text{ when pursuing goal } g. \tag{3.28}$$

The corresponding goal-conditioned action-value function is defined as

$$Q^\pi(s, a, g) = \mathbb{E}_{\pi,P}\left[\sum_{k=0}^{\infty} \gamma^k R_g(s_{t+k}, a_{t+k}, s_{t+k+1}) \,\middle|\, s_t = s, a_t = a\right], \tag{3.29}$$

and measures the expected return when taking action a in state s while pursuing goal g. Training proceeds analogously to standard RL, but with goals drawn from a distribution $p(g)$ and often with relabeling strategies to improve sample efficiency.

3.2.4 Offline RL

In many real-world applications, collecting new interaction data is expensive, risky, or impractical. In such scenarios, *Offline Reinforcement Learning*, also known as *Batch RL*, enables agents to learn effective policies from a fixed dataset of previously collected experiences, without any additional environment interaction. An offline RL dataset typically consists of reward-free trajectories $\{(s_t, a_t, s_{t+1})\}_{t=0}^{T}$ gathered by one or more behavior policies.

Offline RL algorithms must address challenges such as *distributional shift*, which arises when the learned policy selects actions that are insufficiently represented in the dataset, leading to unreliable and overly optimistic value estimates. To mitigate these issues and ensure robust policy learning from static data, techniques such as conservative value estimation, policy constraints, and uncertainty quantification are commonly employed.

3.2.4.1 Implicit Q-Learning

This is one of the most popular offline RL algorithms, introduced by Kostrikov et al. [13]. The goal is to obtain a policy that maximizes the cumulative discounted returns:

$$J(\pi) = \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1})\right], \tag{3.30}$$

using only a fixed dataset $\mathcal{D} = \{(s_i, a_i, s'_i)\}_{i=1}^{N}$ of transitions. Like many offline RL methods, IQL builds on approximate dynamic programming methods that minimize the temporal difference error, according to the following loss function:

$$L_{TD}(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\Big(r(s, a) + \gamma \max_{a'} Q_\phi(s', a') - Q_\theta(s, a)\Big)^2\right], \tag{3.31}$$

where Q_θ is the action-value function approximator parameterized by θ and Q_ϕ is a target network with parameters ϕ. The policy is then defined as

$$\pi(s) = \arg\max_a Q_\theta(s, a). \tag{3.32}$$

To prevent overestimation of Q_θ on out-of-distribution actions, IQL employs *expectile regression* to approximate the maximum value of the Q-function over the possible actions.

For $\tau \in (0, 1)$ the expectile of a random variable X is defined as the solution to the asymmetric least squares problem:

$$\arg\min_{m_\tau} \mathbb{E}\Big[L_2^\tau(x - m_\tau)\Big], \tag{3.33}$$

where $L_2^\tau(x) = |\tau - \mathbf{1}(x < 0)|x^2$ and $\mathbf{1}(\cdot)$ is the indicator function. When $\tau = 0.5$, this reduces to the standard mean squared error loss. When $\tau > 0.5$, the loss penalizes the contributions of values below m_τ more heavily, effectively pushing the estimate upwards.

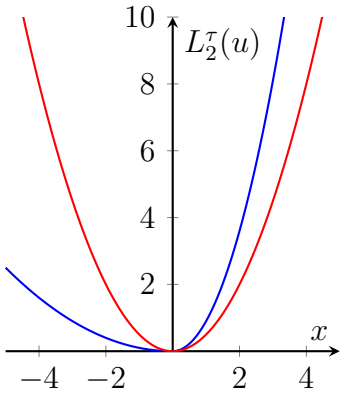


Figure 2: Expectile regression for different values of τ: 0.5 (red), 0.9 (blue)

We can then modify the policy evaluation objective in equation (3.31) to predict an upper expectile of the TD targets that approximates the maximum of $r(s, a) + \gamma Q_\phi(s', a')$ over a'. The modified loss becomes:

$$L(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\Big[L_2^\tau\Big(r(s, a) + \gamma Q_\phi(s', a') - Q_\theta(s, a)\Big)\Big]. \tag{3.34}$$

The idea of directly applying expectile regression to the TD targets, however, has an important limitation. The quantity $r(s, a) + \gamma Q_\phi(s', a')$ depends not only on the actions observed in the dataset, but also on randomness introduced by the environment dynamics $s' \sim p(\cdot \mid s, a)$. As a consequence, a single high-valued transition may inflate the expectile estimate even if no action consistently achieves such a good outcome. In other words, the target may reflect a "lucky" next state rather than a reliably good action.

To separate this stochasticity from the estimation of action values, IQL introduces an additional value function V_ψ that captures an expectile of $Q_\theta(s,a)$ *only with respect to the actions present in the dataset*. This is obtained by solving the regression problem

$$L_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[L_2^\tau(Q_{\hat{\theta}}(s,a) - V_\psi(s))\right], \tag{3.35}$$

where $Q_{\hat{\theta}}$ is a lagged or target copy of the Q-function.

The resulting estimate $V_\psi(s)$ provides a more reliable baseline for policy evaluation. Using it, the Q-function can be updated through a simple squared TD error that averages over the stochastic transitions:

$$L_Q(\theta) = \mathbb{E}_{(s,a,s')\sim\mathcal{D}}\left[\Big(r(s,a) + \gamma V_\psi(s') - Q_\theta(s,a)\Big)^2\right]. \tag{3.36}$$

Both training objectives operate exclusively on actions observed in the dataset, removing the need to maximize over unseen actions and preventing the Q-function from being evaluated on out-of-distribution inputs.

Relevance to World-Model-Based Planning Goal-conditioned value functions such as those learned by IQL provide a natural complement to world models for navigation tasks. While world models enable forward simulation and trajectory optimization, a learned value function can serve as a more topology-aware cost signal than simple Euclidean distance, particularly in environments with obstacles. We discuss this connection further in chapter 5.

Chapter 4

Joint Embedding Predictive Architecture

Traditional world models often aim to predict future observations directly in the input space. A prominent example is Large Language Models (LLMs), which predict the next word in a sequence given the previous ones. Despite their success, such models struggle when the data is continuous, high-dimensional, and complex, as in videos. Predicting every pixel is computationally expensive and can lead to inefficiencies. Moreover, visual data often contain large amounts of information that are irrelevant to the task at hand, making it difficult for the model to focus on the features that truly matter. To address these challenges, *Joint-Embedding Predictive Architectures* (JEPA) [14] adopt a different approach: instead of predicting raw observations, they learn to predict *representations* that capture the essential and predictable features of the data.

4.1 Architecture Overview

A JEPA model consists of three main components: an *encoder*, a *predictor*, and, optionally, a *decoder* (see figure 1). The decoder is not strictly necessary for the model to function as a world model, but it can be included to enable visualization and qualitative analysis of the learned representations and predicted future states.

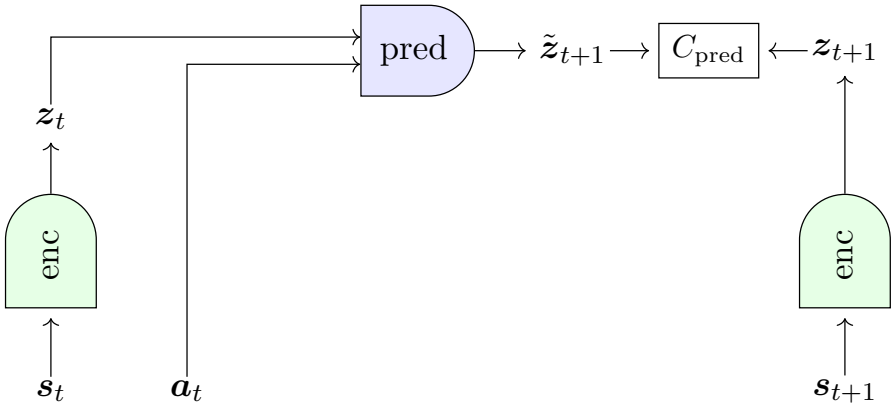


Figure 1: Architecture of a JEPA model. The encoder maps input observations to latent representations, the predictor forecasts future latent states based on current representations and the action taken. The prediction cost is computed between the predicted and true latent representations

4.1.1 Encoder

The encoder is responsible for mapping input observations and actions into a representation space. This is achieved using neural networks that process the input data and extract relevant features while discarding irrelevant or non-predictable information. The encoder can be pretrained using the techniques discussed in chapter 2 to achieve a good initial representation of the input data. However, to fully embrace the JEPA paradigm, the encoder should be fine-tuned jointly with the predictor, allowing it to learn which parts of the information are predictable and relevant for the task.

To better understand what the meaning of *predictable information* is, consider the following example. Suppose you are watching a tree swaying in the wind. Now imagine you can control the wind, and you decide to blow harder from right to left. Can you predict the exact position of every leaf on the tree in the next frame? Probably not. The motion of individual leaves is influenced by many small-scale factors such as turbulence and microscopic air currents. However, you can predict the overall motion of the tree: it will bend toward the left. Thus, the encoder should learn to focus on the large-scale motion of the tree and ignore the small-scale, unpredictable details of individual leaves.

We denote the state encoder parameterized by $\boldsymbol{\theta}$ as $\text{enc}_{\boldsymbol{\theta}}(\cdot)$. It maps an input observation $\boldsymbol{s}_t \in \mathcal{S}$ to a representation $\boldsymbol{z}_t \in \mathcal{Z}$:

$$\boldsymbol{z}_t = \text{enc}(\boldsymbol{s}_t) = \text{enc}_{\boldsymbol{\theta}}(\boldsymbol{s}_t). \tag{4.1}$$

The representation space $\mathcal{Z}$ is typically assumed to be a subset of $\mathbb{R}^d$ for some embedding dimension d.

Figure 2: Tree before and after a gust of wind. While the precise motion of individual leaves is difficult to predict, the overall motion of the tree can be anticipated based on the wind direction.

4.1.2 Predictor

The predictor models the temporal dynamics of the environment in the representation space. It takes as input the encoded representations of the current state and action and predicts the representation of a future state. Formally, the predictor parameterized by $\boldsymbol{\phi}$ is defined as

$$\text{pred}_\phi : \mathcal{Z} \times \mathcal{A} \to \mathcal{Z}, \tag{4.2}$$

which maps the current latent state $\boldsymbol{z}_t$ and the corresponding latent action $\boldsymbol{a}_t$ to a predicted future latent state $\tilde{\boldsymbol{z}}_{t+1}$:

$$\tilde{\boldsymbol{z}}_{t+1} = \text{pred}_\phi\Big(\boldsymbol{z}_t, \boldsymbol{a}_t\Big). \tag{4.3}$$

Depending on the temporal nature of the task, the predictor can be implemented using various neural architectures, such as recurrent networks (RNNs), transformers, or simple feedforward networks. Moreover, the predictor can perform multi-step predictions by recursively applying itself over a sequence of encoded actions:

$$\tilde{\boldsymbol{z}}^s_{t+k} = \text{pred}^{(k)}_\phi\Big(\boldsymbol{z}^s_t, \boldsymbol{z}^a_{t:t+k-1}\Big), \tag{4.4}$$

where $\text{pred}^{(k)}_\phi$ denotes k recursive applications of the predictor. This allows the model to perform rollouts in representation space, sharing parameters across time steps and enabling long-term planning and decision-making. An important fact to keep in mind is that any error in the prediction will compound over time, leading to an exponential divergence from the true trajectory.

4.1.3 Decoder

The decoder is used to project the state representations back into the input space. It can be trained jointly with the encoder and predictor or separately, depending on the encoder training strategy. The decoder may not necessarily reconstruct in the same input space. For example, if a state contains both internal variables and a visual observation, the decoder may be trained to reconstruct different targets: the internal

state alone, the full image, a low-resolution or grayscale version of the image, or any alternative representation that is simpler to decode but remains informative.

We denote the decoder, parameterized by $\boldsymbol{\psi}$, as $\text{dec}_{\boldsymbol{\psi}}(\cdot)$, which maps a representation $\boldsymbol{z}_t^s$ back to the input space, producing a reconstructed observation $\tilde{\boldsymbol{s}}_t$:

$$\tilde{\boldsymbol{s}}_t = \text{dec}_{\boldsymbol{\psi}}(\boldsymbol{z}_t^s) \tag{4.5}$$

4.2 Training

Training a JEPA model involves optimizing the encoder and predictor to minimize a cost function that measures the discrepancy between the predicted and true future representations. This is a regression problem in the representation space, and consequently the cost function to minimize is typically the squared Euclidean distance. Again, the decoder can be trained jointly or separately. It might be trained together with the encoder to form an autoencoder, or it might be used for regularization purposes.

Let $\boldsymbol{s}_t$ be the current observation at time step t, $\boldsymbol{a}_t$ the action taken at time step t, and $\boldsymbol{s}_{t+1}$ be the subsequent observation at time step $t+1$. The encoder produces the representations:

$$\boldsymbol{z}_t = \text{enc}_{\boldsymbol{\theta}}(\boldsymbol{s}_t), \tag{4.6}$$

$$\boldsymbol{z}_{t+1} = \text{enc}_{\boldsymbol{\theta}}(\boldsymbol{s}_{t+1}). \tag{4.7}$$

The predictor then forecasts the next representation:

$$\tilde{\boldsymbol{z}}_{t+1} = \text{pred}_{\boldsymbol{\phi}}(\boldsymbol{z}_t, \boldsymbol{a}_t). \tag{4.8}$$

We define the *prediction cost* as the Euclidean distance between the predicted representation and the true representation of the next observation:

$$C_{\text{pred}}(\tilde{\boldsymbol{z}}_{t+1}, \boldsymbol{z}_{t+1}) = \|\tilde{\boldsymbol{z}}_{t+1} - \boldsymbol{z}_{t+1}\|^2. \tag{4.9}$$

This apparently simple setup brings challenges of the same nature as those encountered in the encoder-only models discussed in chapter 2. In particular, there is a risk of *representation collapse*: if the model minimizes only the prediction cost, the encoder may learn to map all inputs to a constant vector $\boldsymbol{c}$, effectively collapsing the representation space. At the same time, the predictor can trivially learn to ignore the action taken, and apply the identity function to the representation, driving the loss to zero without learning any meaningful dynamics.

$$\text{enc}_{\boldsymbol{\theta}_s}(\boldsymbol{s}_t) = \boldsymbol{c}, \quad \forall \boldsymbol{s}_t \in \mathcal{S}, \tag{4.10}$$

$$\text{pred}_{\boldsymbol{\phi}}(\boldsymbol{c}, \boldsymbol{a}_t) = \boldsymbol{c}, \quad \forall \boldsymbol{a}_t \in \mathcal{A}, \tag{4.11}$$

$$C_{\text{pred}}(\tilde{\boldsymbol{z}}_{t+1}^s, \boldsymbol{z}_{t+1}^s) = \|\boldsymbol{c} - \boldsymbol{c}\|^2 = 0. \tag{4.12}$$

To prevent such collapse and to enforce meaningful latent dynamics, a JEPA model must adopt some of the strategies discussed in chapter 2. These include architectural choices, such as using asymmetric networks for the encoder and predictor, as well as regularization techniques that encourage diversity, temporal coherence, and causally relevant representations.

4.2.1 Regularization Techniques

Sobal *et al.* [23] propose a combination of three regularization techniques to prevent representation collapse:

- A VICReg-inspired objective to encourage diversity along the time dimension to capture information that changes
- An Inverse Dynamics Model (IDM) regularization, to ensure the latent representation retains causally relevant aspects of the environment.
- A tunable objective to enforce temporal smoothness of learned representations

VICReg-inspired Objective Let $\boldsymbol{z} \in \mathbb{R}^{H\times N\times D}$ denote the matrix of latent representations obtained by encoding a batch of N trajectories of horizon H, each represented in a D-dimensional latent space. Following [4], the invariance term is dropped and only the regularization terms are preserved:

- **Variance cost:** This is a hinge cost that encourages the encoder and predictor to capture features that vary over time:

$$C_{\text{var}}(\boldsymbol{z}) = \frac{1}{HD}\sum_{h=0}^{H-1}\sum_{d=1}^{D}\max\Big(0, \gamma - \sqrt{\text{Var}(z_{h,:,d}) + \varepsilon}\Big),$$

  where $\gamma > 0$, usually set to 1, ensures that each latent dimension has nonzero variance, and ε is a small constant for numerical stability, since the gradient of the square root becomes unbounded near zero.
- **Covariance cost:** This term avoids redundant features by penalizing correlations between different latent dimensions:

$$C_{\text{cov}}(\boldsymbol{z}) = \frac{1}{HD}\sum_{h=0}^{H-1}\sum_{i\neq j}\Big[\boldsymbol{\Sigma}(z_h)\Big]_{i,j}^2, \quad \boldsymbol{\Sigma}(z_h) = \frac{1}{N-1}(z_h - \bar{z}_h)^\top(z_h - \bar{z}_h), \tag{4.13}$$

  where $\bar{z}_h$ is the mean of z_h along the batch dimension.

The combined VCReg regularization term is then

$$L_{\text{VCReg}} = \lambda_{\text{var}} C_{\text{var}} + \lambda_{\text{cov}} C_{\text{cov}}, \tag{4.14}$$

where $\lambda_{\text{var}}, \lambda_{\text{cov}}$ are tunable coefficients.

Inverse Dynamics Regularization To further prevent degenerate latent dynamics and ensure that the predictor utilizes the action information, an auxiliary inverse dynamics model $\text{IDM}_{\boldsymbol{\omega}}$ is trained to infer the action $\boldsymbol{a}_t$ from consecutive latent states $(\boldsymbol{z}_t, \boldsymbol{z}_{t+1})$:

$$C_{\text{IDM}}(\boldsymbol{a}_t, \boldsymbol{z}_t, \boldsymbol{z}_{t+1}) = \frac{1}{HN}\sum_{h=0}^{H-1}\sum_{n=1}^{N}\Big\|\boldsymbol{a}_{h,n} - \text{IDM}_{\boldsymbol{\omega}}(\boldsymbol{z}_{h,n}, \boldsymbol{z}_{h+1,n})\Big\|_2^2. \tag{4.15}$$

This auxiliary cost provides a reconstruction signal in the *action space*, encouraging the latent dynamics to retain information about the control inputs that caused each transition. Without this constraint, if the precollected trajectories lack sufficient diversity (e.g., nearly identical actions are taken in a given state), the predictor may learn to evolve latent states independently of the actions. The inverse dynamics regularization mitigates this issue by explicitly enforcing that consecutive latent states encode enough information to *reconstruct* the action responsible for their transition.

Temporal Smoothness To encourage local smoothness in the learned latent space, a temporal smoothness cost is applied:

$$C_{\text{smooth}} = \frac{1}{HN} \sum_{h=0}^{H-2} \sum_{n=1}^{N} \|\boldsymbol{z}_{h+1,n} - \boldsymbol{z}_{h,n}\|_2^2. \tag{4.16}$$

This regularization discourages abrupt changes in the latent trajectory and biases the encoder toward representing slowly varying and predictable features [22]. This can be seen as the invariance term from VICReg, extracted and applied separately over time.

Total Training Objective The overall training loss of the JEPA-based world model combines the prediction cost with the above regularizers:

$$L_{\text{total}} = C_{\text{pred}} + \alpha_{\text{VCReg}} L_{\text{VCReg}} + \alpha_{\text{IDM}} L_{\text{IDM}} + \alpha_{\text{time}} L_{\text{time}}, \tag{4.17}$$

where the coefficients α control the contribution of each term. The authors performed an ablation study to assess the impact of each regularization component and found that the VCReg term is crucial to prevent representation collapse, while the IDM and temporal smoothness regularizations further enhance the quality of the learned dynamics and representations.

4.2.2 Teacher-Student Architectures

Another effective and simpler way to prevent representation collapse is to use a *teacher-student* BYOL-like architecture. In this setup, two instances of the same encoder network are maintained: a *student* network that is updated via gradient descent, and a *teacher* network that is updated using an exponential moving average (EMA) of the student's parameters. The predictor is trained to predict the latent representations produced by the teacher network, which provides a stable target for learning. This approach has been shown to effectively prevent collapse without the need for additional regularization terms.

$$C_{\text{pred}}(\tilde{\boldsymbol{z}}_{t+1}, \boldsymbol{z}_{t+1}) = \|\text{pred}(\text{enc}_{\text{stud}}(\boldsymbol{s}_t), \boldsymbol{a}_t) - \text{sg}(\text{enc}_{\text{teach}}(\boldsymbol{s}_t))\|^2, \tag{4.18}$$

$$\boldsymbol{\theta}_{\text{teach}} \leftarrow \tau \boldsymbol{\theta}_{\text{teach}} + (1-\tau)\boldsymbol{\theta}_{\text{stud}}, \tag{4.19}$$

where $\text{sg}(\cdot)$ denotes the stop-gradient operation, and $\tau \in [0, 1)$ is the EMA decay rate.

This strategy has been widely adopted as a collapse-prevention mechanism in self-supervised learning. It was first introduced in BYOL [9] and subsequently analyzed empirically [7] and theoretically [24]. The same principle underlies several JEPA-style models, including *I-JEPA* [14], *DINO-WM* [28], and the video-based *V-JEPA* and *V-JEPA 2* [5, 2]. In *V-JEPA 2*, the authors further modify the prediction objective to use an ℓ_1 regression loss, finding it to stabilize training and improve convergence compared to the conventional ℓ_2 norm. Hybrid losses as the Huber loss have also been explored in this context.

The effectiveness of this approach can be interpreted as follows. Let $Y = \mathrm{sg}(\mathrm{enc}_{\mathrm{teach}}(\boldsymbol{s}_t))$ denote the target latent representation produced by the teacher encoder, P the predictor, and $E_\theta = \mathrm{enc}_{\mathrm{stud}}$. The optimal predictor satisfies

$$P^\star(E_\theta(\boldsymbol{s}_t)) = \arg\min_P \|P(E_\theta(\boldsymbol{s}_t)) - Y\|_1 = \mathrm{median}(Y \mid E_\theta(\boldsymbol{s}_t)). \tag{4.20}$$

Substituting this expression into the prediction cost and taking the expected gradient with respect to the encoder yields

$$\nabla_\theta \mathbb{E}[\|P^\star(E_\theta(\boldsymbol{s}_t)) - Y\|_1] = \nabla_\theta \,\mathrm{MAD}(Y \mid E_\theta(\boldsymbol{s}_t)), \tag{4.21}$$

where $\mathrm{MAD}(\cdot)$ denotes the median absolute deviation. Thus, when the predictor is optimal, the encoder must learn to capture as much information as possible from the video to minimize its deviation from the target representation. The underlying hypothesis is that maintaining the target representation through an EMA of the encoder parameters allows the predictor to evolve faster than the encoder, remaining close to optimal and thereby preventing representational collapse.

4.3 Hierarchical JEPA

In this last section we present the idea behind *Hierarchical JEPA* (H-JEPA) [14, 15], which extends the JEPA framework to multiple levels of abstraction.

4.3.1 Motivation

While a single-level JEPA can learn predictive representations of the world, it remains limited to a single temporal and spatial scale. Real-world environments, however, exhibit structure across multiple levels of abstraction: low-level sensory dynamics (e.g., pixels, joint angles) evolve much faster than high-level latent causes such as object configurations, goals, or intentions.

In real life, humans and animals naturally process information at multiple layers of abstraction. For instance, if a person wants to go to the grocery store, they might first plan the high-level route (walk to the car, drive to the store), then execute mid-level actions (dressing, walking, driving), and finally perform low-level motor commands (moving legs, steering the wheel). This hierarchical structure allows for efficient planning and decision-making, as high-level goals can be decomposed into manageable sub-tasks.

To efficiently capture such multi-scale regularities, *Hierarchical JEPA (H-JEPA)* extends the original architecture by stacking multiple JEPA modules operating at progressively coarser spatial or temporal resolutions. Each layer encodes information relevant to its scale, forming a predictive hierarchy of latent variables reminiscent of cortical processing hierarchies in biological systems.

4.3.2 Architecture

An H-JEPA is composed of L levels, indexed by $l = 1, \ldots, L$. Each level consists of an encoder $\text{enc}^{(l)}(\cdot)$, a predictor $\text{pred}^{(l)}(\cdot)$, and optionally a decoder $\text{dec}^{(l)}(\cdot)$. The input to the first level is the raw sensory observation $\boldsymbol{x}_t$, while higher levels operate on latent representations from the level below:

$$\boldsymbol{z}_t^{(l)} = \text{enc}^{(l)}\Big(\boldsymbol{z}_{t-\Delta_l:t+\Delta_l}^{(l-1)}\Big), \quad \boldsymbol{z}_t^{(0)} = \boldsymbol{x}_t. \tag{4.22}$$

Each level learns to predict future representations at its own temporal scale:

$$\tilde{\boldsymbol{z}}_{t+T_l}^{(l)} = \text{pred}^{(l)}\Big(\boldsymbol{z}_t^{(l)}, \boldsymbol{a}_t\Big), \tag{4.23}$$

where T_l is the prediction horizon for level l, and $\boldsymbol{a}_t$ is the action taken at time t. The optional decoder reconstructs inputs from the latent space:

$$\tilde{\boldsymbol{z}}_t^{(l-1)} = \text{dec}^{(l)}\Big(\boldsymbol{z}_t^{(l)}\Big). \tag{4.24}$$

Each level is trained to minimize a prediction loss similar to that of the single-level JEPA, along with any necessary regularization terms to prevent collapse. The total loss is a weighted sum of the losses from all levels:

$$L_{\text{total}} = \sum_{l=1}^{L} \alpha_l L^{(l)}, \tag{4.25}$$

where α_l are tunable coefficients.

Chapter 5

Implementation and Experiments

In this chapter, we describe the implementation details of our proposed method. All experiments were performed using Python 3.12 and PyTorch 2.1.0.

5.1 Environment

We adopt the PointMaze environment from the `gymnasium-robotics` suite [25] as our primary testbed for evaluating the JEPA-based agent. The environment consists of a 2D point-mass agent that navigates through maze layouts of varying complexity to reach a designated goal. The agent's internal state, or proprioceptive state, is defined as

$$\boldsymbol{s}_t = (x_t, y_t, v_t^x, v_t^y) \in \mathbb{R}^4, \tag{5.1}$$

where (x_t, y_t) denotes the agent's position and (v_t^x, v_t^y) its velocity. Importantly, the agent exhibits *momentum*: dynamics are second-order in position, as actions affect acceleration, which then updates velocities and positions over time.

At each timestep the agent performs a continuous 2D action

$$\boldsymbol{a}_t \in [-1, 1]^2, \tag{5.2}$$

representing forces applied along the x- and y-axes. Velocities are clipped to $\pm 5\,\mathrm{m/s}$ in accordance with the official implementation. The transition dynamics follow a point-mass model with friction:

$$\boldsymbol{v}_{t+1} = \mathrm{clip}\Big(\boldsymbol{v}_t + \Delta t\, \kappa\, \boldsymbol{a}_t - \mu\, \boldsymbol{v}_t,\ [-v_{\max}, v_{\max}]^2\Big), \tag{5.3}$$

$$\boldsymbol{p}_{t+1} = \boldsymbol{p}_t + \Delta t \cdot \boldsymbol{v}_{t+1}, \tag{5.4}$$

after which the position is projected onto the feasible maze region. Collisions clamp the next position and reset tangential velocity components, causing a sliding-along-walls behaviour. Because of momentum, the agent cannot instantly stop or change direction, which makes navigation substantially more challenging.

Although the true state contains privileged coordinates (x_t, y_t), the JEPA agent in our experiments does not observe them. Instead, the agent receives:

- a rendered RGB image of the current maze state, $o_t \in \mathbb{R}^{64\times64\times3}$,
- its instantaneous velocity (v_t^x, v_t^y).

This setting forces the representation to infer spatial structure, localization, and motion dynamics directly from images and velocity. The environment exposes a goal position $\boldsymbol{g} = (x^\star, y^\star)$; episodes terminate upon reaching a neighbourhood of the goal or after a maximum horizon.

5.2 Data

Each dataset consists of 10 000 trajectories of length 100. The agent interacts with 40 distinct maze layouts for training and another 40 disjoint layouts for testing, ensuring evaluation on unseen configurations. Some sample maze layouts are shown in figure 1.

To obtain diverse and exploratory trajectories we employ a *stochastic policy* based on the *von Mises distribution*, a circular analogue of the normal distribution commonly used to model directional data. At each timestep t, the action direction θ_t is sampled from a von Mises distribution centered at the previous direction θ_{t-1} with concentration parameter κ. Formally,

$$\theta_t \sim \text{von Mises}(\mu = \theta_{t-1}, \kappa) \tag{5.5}$$

whose probability density function is given by

$$f_{\mu,\kappa}(\theta) = \frac{e^{\kappa\cos(\theta-\mu)}}{2\pi I_0(\kappa)}, \tag{5.6}$$

where $I_0(\kappa)$ is the modified Bessel function of the first kind of order zero. Higher values of κ correspond to stronger directional persistence (i.e., smoother, more coherent trajectories), while smaller values induce more random motion. In practice, the authors set $\kappa = 5$, yielding trajectories that balance randomness and smooth exploration across the maze [23]. The action magnitude is then sampled uniformly within a fixed bound (up to 2.45 in the original experiments), and the resulting 2D displacement vector is applied to update the agent's position.

This stochastic controller generates realistic yet non-optimal motion patterns, producing the type of suboptimal reward-free trajectories necessary for training and evaluating world-model-based agents.

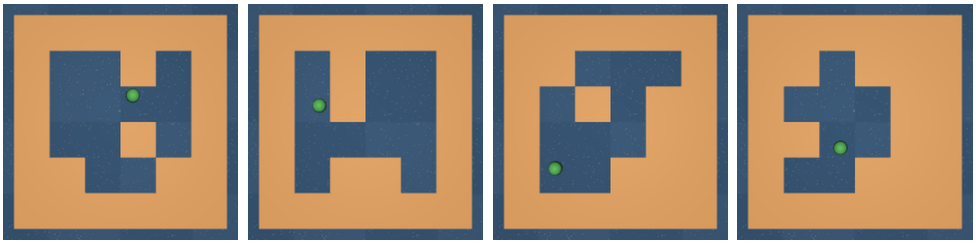


Figure 1: Sample maze layouts from the PointMaze environment.

5.3 Architecture

The JEPA model comprises several neural networks and is implemented using the PyTorch framework [19]. Hereafter we detail the architecture of each component.

Encoder We have three encoders in total, one for the visual data, one for the proprioceptive data (velocity), and one for the action.

The visual encoder network maps raw RGB observations to latent representations. The architecture is a variant of MeNet6, a four-layer CNN in which we modify the second convolutional layer to use stride 1 instead of 2, and add an average pooling layer afterward to obtain smoother representations. This encoder maps $3 \times 64 \times 64$ RGB images to $16 \times 26 \times 26$ feature maps and has approximately 28 k parameters.

Layer	Output Channels	Details
$\text{Conv}_{5\times5}$ + GN(4) + ReLU	16	stride 1, no padding
$\text{Conv}_{5\times5}$ + GN(8) + ReLU	32	stride 1, no padding
$\text{AvgPool}_{2\times2}$	32	stride 2
$\text{Conv}_{3\times3}$ + GN(8) + ReLU	32	stride 1, no padding
$\text{Conv}_{3\times3}$	16	stride 1, padding 1

Table 5.1: Architecture of the visual encoder *SmoothMeNet6*.

The proprioceptive encoder is a simple one dimensional batchnorm layer followed by a 2D expansion that maps the velocity vector to a $2 \times 26 \times 26$ static feature map. In total, the proprioceptive encoder has 4 parameters.

The action encoder is exactly the same as the proprioceptive encoder, mapping the 2D action vector to a $2 \times 26 \times 26$ static feature map. In total, the action encoder has 4 parameters.

Mask extractor The mask extractor is a two layers CNN that takes as input the encoded visual representation and outputs a $1 \times 26 \times 26$ mask with sigmoid applied component-wise. The mask is then used to separate the visual latent representation into static and dynamic parts.

$$\boldsymbol{M}_t = \sigma\left(\text{MaskExtractor}(\boldsymbol{z}_t^{vis})\right) \tag{5.7}$$

$$\boldsymbol{z}_t^{vis,dyn} = \boldsymbol{z}_t^{vis} \odot \boldsymbol{M}_t \tag{5.8}$$

$$\boldsymbol{z}_t^{vis,stat} = \boldsymbol{z}_t^{vis} \odot (1 - \boldsymbol{M}_t) \tag{5.9}$$

The purpose of this module is to help the model disentangle static features (e.g., walls, floor texture) from dynamic features (e.g., agent position, moving objects) in the visual

input. The idea is that two consecutive states can be decomposed as

$$\boldsymbol{z}_t^{vis} = \boldsymbol{z}_t^{vis,stat} + \boldsymbol{z}_t^{vis,dyn} \tag{5.10}$$

$$\boldsymbol{z}_{t+1}^{vis} = \boldsymbol{z}_{t+1}^{vis,stat} + \boldsymbol{z}_{t+1}^{vis,dyn} \tag{5.11}$$

then the difference between the two states can be expressed as

$$\boldsymbol{z}_{t+1}^{vis} - \boldsymbol{z}_t^{vis} = (\boldsymbol{z}_{t+1}^{vis,dyn} - \boldsymbol{z}_t^{vis,dyn}) + (\boldsymbol{z}_{t+1}^{vis,stat} - \boldsymbol{z}_t^{vis,stat}) \tag{5.12}$$

If the static part remains constant over time, $\boldsymbol{z}_{t+1}^{vis,stat} \approx \boldsymbol{z}_t^{vis,stat}$, then the change in the visual representation is primarily due to the dynamic part:

$$\boldsymbol{z}_{t+1}^{vis} - \boldsymbol{z}_t^{vis} \approx \boldsymbol{z}_{t+1}^{vis,dyn} - \boldsymbol{z}_t^{vis,dyn} \tag{5.13}$$

Layer	Output Channels	Details
$\text{Conv}_{3\times3}$ + GN(4) + ReLU	32	padding 1, stride 1
$\text{Conv}_{1\times1}$	1	stride 1, no padding

Table 5.2: Architecture of the mask extractor.

This separation can improve the model's ability to predict future states by focusing only on the dynamic components that change over time, while ignoring the static background. The extractor has approximately 4.7 k parameters.

Predictor The predictor network forecasts future states and is implemented as a three layers CNN, plus a stabilization layer after the residual connection. Thanks to the mask extractor, the predictor only needs to predict the dynamic part of the visual representation in a residual manner, simplifying the learning task. The predictor function can be expressed as

$$\tilde{\boldsymbol{z}}_{t+1} = \text{Predictor}\left(\boldsymbol{z}_t^{vis,stat}, \boldsymbol{z}_t^{vis,dyn}, \boldsymbol{z}_t^{prop}, \boldsymbol{a}_t\right) \tag{5.14}$$

$$\begin{bmatrix} \tilde{\boldsymbol{z}}_{t+1}^{vis} \\ \tilde{\boldsymbol{z}}_{t+1}^{prop} \end{bmatrix} = \text{Stabilize}\left(\begin{bmatrix} \boldsymbol{z}_t^{vis,stat} \\ \mathbf{0} \end{bmatrix} + \text{PredictorNet}\left(\boldsymbol{z}_t^{vis,dyn}, \boldsymbol{z}_t^{prop}, \boldsymbol{a}_t\right)\right) \tag{5.15}$$

where PredictorNet$(\cdot)$ is the CNN that predicts the changes in the dynamic visual and proprioceptive representations, and Stabilize$(\cdot)$ is a single convolutional layer that stabilizes the output of the predictor. The inputs to the predictor net are concatenated along the channel dimension, resulting in a $20 \times 26 \times 26$ feature map, and the output is a $18 \times 26 \times 26$ feature map. The predictor has approximately 33 k parameters.

Layer	Output Channels	Details
$\text{Conv}_{5\times5}$ + GN(4) + ReLU	32	padding 2, stride 1
$\text{Conv}_{3\times3}$ + GN(4) + ReLU	32	padding 2, stride 1
$\text{Conv}_{3\times3}$	18	padding 1, stride 1
$\text{Conv}_{3\times3}$	18	padding 1, stride 1

Table 5.3: Architecture of the convolutional predictor (*ConvPredictor*) and its stabilization layer.

Inverse Dynamics Model The IDM predicts the action taken between two consecutive latent states, ensuring that action information remains useful for forecasting future states. This is especially important when the pre-collected trajectories do not cover the state-action space uniformly. In some regions of the state space, most observed transitions may be associated with nearly identical actions. In such cases, the predictor might exploit this bias and learn a form of "teleportation", effectively ignoring the action input and assuming that the next state is always the same whenever the agent enters that region.

It is implemented as a three-layer CNN that takes as input the concatenated latent representations of the current and next visual and proprioceptive data, and outputs the action taken for the transition to occur. The IDM function can be expressed as

$$\tilde{\boldsymbol{a}}_t = \text{IDM}\left(\boldsymbol{z}_t, \boldsymbol{z}_{t+1}\right) \tag{5.16}$$

However, since we are decoupling static and dynamic features in the visual representation, we only provide the dynamic part of the visual representation to the IDM. The inputs are concatenated along the channel dimension, resulting in a $36 \times 26 \times 26$ feature map, and the output is a $2 \times 26 \times 26$ feature map. The IDM has approximately 22 k parameters.

Layer	Output Channels	Details
$\text{Conv}_{5\times5}$ + GN(8) + ReLU	24	padding 2, stride 1
$\text{Conv}_{3\times3}$	1	padding 1, stride 1
LayerNorm + Linear + ReLU	64	flatten (26×26) input
Linear	2	output action components

Table 5.4: Architecture of the inverse dynamics decoder (*IDMDecoderConv*).

5.4 Training

In order to train the JEPA model, we introduce several objectives that encourage the learning of meaningful representations and accurate predictions of future states. Moreover, we adopt a stopgrad strategy similar to SimSiam [7] to prevent collapse during training. This means the gradient does not flow through the target representations when computing the prediction loss, but only through the predicted ones.

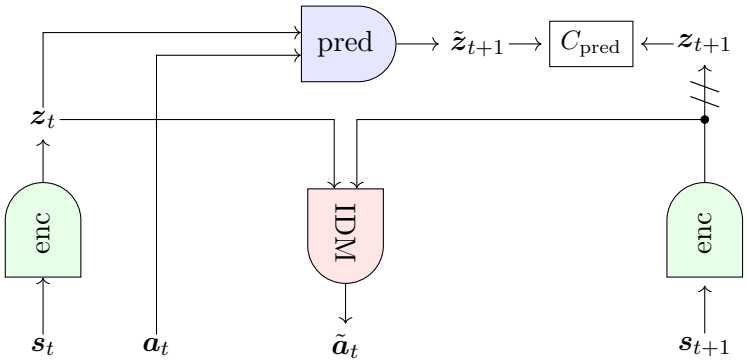


Figure 2: JEPA training diagram. The encoded state and action are used by the predictor to forecast future states. The IDM module predict the taken action from the two consecutive states. Stopgrad is applied to the target latent representations when computing the prediction loss. The masking mechanism is not shown for clarity and assumed to be part of the predictor.

5.4.1 Objectives

We introduce several training objectives to effectively learn the JEPA model within the PointMaze environment. These can be identified as prediction cost and regularization costs. The first focuses on ensuring accurate predictions of future latent states, while the second prevents information collapse and encourages the learning of useful representations.

Prediction Cost We consider the Mean Squared Error (MSE) between the predicted future latent states and the actual future latent states. This cost encourages the model to accurately predict the future observations based on the current state and action.

$$C_{\text{pred}}(\boldsymbol{z}_t, \tilde{\boldsymbol{z}}_t) = \mathbb{E}\left[\|\boldsymbol{z}_{t+1} - \tilde{\boldsymbol{z}}_{t+1}\|^2\right] \tag{5.17}$$

where $\boldsymbol{z}_{t+1}$ is the actual latent state at time $t+1$ and $\tilde{\boldsymbol{z}}_{t+1}$ is the predicted latent state. Since we have access to entire subsequences of length H during training, we perform autoregressive predictions over the whole horizon and average the loss across all timesteps with a decay factor $\gamma \in (0, 1]$:

$$C_{\text{pred}}(\tilde{\boldsymbol{z}}_{t:t+H}, \boldsymbol{z}_{t:t+H}) = \frac{1}{H}\frac{1-\gamma}{1-\gamma^{H-1}}\sum_{h=1}^{H}\gamma^{h-1}\mathbb{E}\left[\|\boldsymbol{z}_{t+h} - \tilde{\boldsymbol{z}}_{t+h}\|^2\right] \tag{5.18}$$

This encourages accurate and stable long-term predictions while prioritizing near-term accuracy and is particularly useful for our planning purposes. In all our experiments we set $\gamma = 0.5$ and $H = 8$.

Inverse Dynamics Modelling (IDM) We consider the MSE between the actual action taken and the action predicted by the IDM module.

$$C_{\text{IDM}}(\boldsymbol{a}_t, \tilde{\boldsymbol{a}}_t) = \mathbb{E}\left[\|\boldsymbol{a}_t - \tilde{\boldsymbol{a}}_t\|^2\right] \tag{5.19}$$

where $\boldsymbol{a}_t$ is the actual action taken at time t and $\tilde{\boldsymbol{a}}_t$ is the predicted action from the IDM module.

Temporal Invariance To encourage the model to disentangle static and dynamic features in the visual input, we introduce a loss that penalizes changes in the static part of the visual representation over time.

$$C_{\text{static}}(\boldsymbol{z}_t^{vis,static}, \boldsymbol{z}_{t+1}^{vis,static}) = \mathbb{E}\left[\|\boldsymbol{z}_{t+1}^{vis,static} - \boldsymbol{z}_t^{vis,static}\|^2\right] \tag{5.20}$$

This cost, together with the residual prediction mechanism, forces the model to focus on predicting only the dynamic components of the scene, avoiding unnecessary changes to the static background.

Mask regularization In order to make our mask extractor learn a meaningful separation between static and dynamic features, we need to prevent some degenerate solutions.

The first degenerate solution for the mask extractor is to output a mask of all zeros. This would draw the temporal invariance loss to zero, but would prevent the predictor from distinguishing between static and dynamic features. To prevent this, we introduce a hinge loss that penalizes the mask for having less than a given percentage of active pixels.

$$C_{\text{mask\_area}}(\boldsymbol{M}_t) = \mathbb{E}\left[\max(0, \delta - \text{mean}(\boldsymbol{M}_t))\right] \tag{5.21}$$

In all our experiments we set $\delta = 0.03$, meaning that at least 3% of the pixels should be active.

The second undesirable solution is one where the mask fails to identify regions that exhibit temporal variation. When the agent moves, the corresponding visual features should change over time, and the mask is expected to highlight these areas. To enforce this, we penalize cases where temporal change is large but the mask value is small:

$$C_{\text{mask\_alignment}} = \mathbb{E}\Big[(1 - \boldsymbol{M}_t)\,\|\boldsymbol{z}_{t+1}^{\text{vis}} - \boldsymbol{z}_t^{\text{vis}}\|^2\Big]\,. \tag{5.22}$$

A third failure occurs when the mask becomes spatially fragmented, resulting in scattered activations rather than a compact region corresponding to the agent. To promote spatial coherence, we introduce a total-variation regularizer that penalizes large local gradients:

$$C_{\text{mask\_compactness}} = \mathbb{E}[\|\nabla_x \boldsymbol{M}_t\|_1 + \|\nabla_y \boldsymbol{M}_t\|_1]\,. \tag{5.23}$$

Total Loss The total training loss is a weighted sum of the individual components:

$$C_{\text{total}} = \lambda_{\text{pred}} C_{\text{pred}} + \lambda_{\text{IDM}} C_{\text{IDM}} + \lambda_{\text{static}} C_{\text{static}} \tag{5.24}$$
$$+ \lambda_{\text{mask\_area}} C_{\text{mask\_area}} \tag{5.25}$$
$$+ \lambda_{\text{mask\_align}} C_{\text{mask\_alignment}} \tag{5.26}$$
$$+ \lambda_{\text{mask\_comp}} C_{\text{mask\_compactness}}. \tag{5.27}$$

The coefficients λ control the relative importance of each term. In all our experiments we use:

$$\lambda_{\text{pred}} = 10.0, \qquad \lambda_{\text{IDM}} = 2.0, \qquad \lambda_{\text{static}} = 1.0, \tag{5.28}$$
$$\lambda_{\text{mask\_area}} = 0.1, \qquad \lambda_{\text{mask\_align}} = 0.1, \qquad \lambda_{\text{mask\_comp}} = 0.1. \tag{5.29}$$

5.4.2 Optimization details

The model is trained using the AdamW optimizer [16]. All parameters are partitioned into two disjoint groups: *encoder* and *predictor*. The predictor group contains only the convolutional predictor network, while the encoder group includes all remaining components (visual encoder, proprioceptive encoder, action encoder, mask extractor, and IDM module). Each group is assigned its own initial learning rate, final learning rate, and weight decay.

Training uses a cosine annealing schedule with a 200-step linear warm-up phase. Warm-up mitigates instabilities during the early optimization steps, stabilizes the running statistics of normalization layers, and allows the adaptive moment estimates in AdamW to converge before reaching the full learning rate.

The optimization hyperparameters are summarized in the table below.

Parameter	Encoder	Predictor
Initial Learning Rate	5×10^{-3}	1×10^{-2}
Final Learning Rate	1×10^{-7}	1×10^{-7}
Weight Decay	1×10^{-5}	1×10^{-5}

Table 5.5: Optimization hyperparameters for the encoder and predictor parameter groups.

The predictor uses a higher initial learning rate than the encoder, reflecting the considerations discussed in Section 4.2, where we argued that the predictor should converge more rapidly to avoid information collapse in the joint embedding.

Using a batch size of 128, the model is trained for 65 300 steps with gradient-norm clipping (`clip_grad_norm`=1.0) to prevent exploding gradients. All experiments are run on a single NVIDIA RTX 4070 Super GPU, and training completes in approximately one hour. To improve training efficiency, we use `bfloat16` mixed precision, which

reduces memory consumption and increases throughput on compatible hardware with negligible impact on model accuracy. We further enable `torch.compile`, which employs TorchDynamo to optimize the computation graph and accelerate execution.

5.5 Planning

Once the JEPA model has been trained, we can use it to perform planning directly in the learned representation space. The masking mechanism plays a crucial role: it enables the construction of an unsupervised coordinate system in which the agent can localize itself and reason about its dynamics. This makes it possible to define a meaningful cost function based on the (Euclidean) distance between the agent's estimated position and the goal. Without masking, distances between visual latent states would not correspond to any geometric structure of the environment.

Extracting position from the mask We obtain an approximate agent position by first thresholding the mask at its 99-th percentile to produce a binary segmentation, and then computing the centroid of the activated pixels:

$$\boldsymbol{M}^{\text{bin}}(\text{enc}(\boldsymbol{s})) = \mathbb{I}[\boldsymbol{M}(\text{enc}(\boldsymbol{s})) > \text{percentile}\left(\boldsymbol{M}(\text{enc}(\boldsymbol{s})), 99\right)], \tag{5.30}$$

$$\tilde{\boldsymbol{p}}(\boldsymbol{s}) = \text{centroid}\Big(\boldsymbol{M}^{\text{bin}}(\text{enc}(\boldsymbol{s}))\Big) = \Big(\tilde{x}(\boldsymbol{s}),\, \tilde{y}(\boldsymbol{s})\Big). \tag{5.31}$$

Although $\tilde{\boldsymbol{p}}(\boldsymbol{s})$ lies in pixel coordinates, it is related to the physical coordinates of the maze by a fixed linear transformation, which preserves the structure needed for planning.

Goal-reaching cost Using this position estimate, we define a simple surrogate cost based on the squared Euclidean distance to the goal:

$$C_{\text{goal}}(\boldsymbol{s}_t, \boldsymbol{s}_g) = \left\|\tilde{\boldsymbol{p}}(\boldsymbol{s}_t) - \tilde{\boldsymbol{p}}(\boldsymbol{s}_g)\right\|_2^2. \tag{5.32}$$

This provides a smooth objective for reaching the goal in the learned coordinate system.

Non-differentiability issue A limitation of this formulation is that the position extraction function is not differentiable because of the thresholding operation. Consequently, C_{goal} is not differentiable with respect to the latent representation, preventing the use of gradient-based trajectory optimizers. In practice, this restricts planning to sampling-based or derivative-free methods.

5.5.1 Optimization

In order to plan a sequence of actions that lead the agent from its current state to a desired goal state, we adopt the Model Predictive Path Integral (MPPI) framework described in section 3.1.3 with the following parameters:

- number of samples: $N = 512$
- planning horizon: $H = 200$
- maximum total timesteps: 400
- temperature: $\lambda = 2.5 \times 10^{-3}$
- action noise standard deviation: $\sigma = 1.0$
- action noise decay: 0.9
- iterations per replanning step: 5
- replanning frequency: variable (from 1 to 10 steps)

The cost function employed is defined as

$$C(\boldsymbol{a}_{1:H}, \boldsymbol{s}_{1:H+1}) = C_{\text{goal}}(\boldsymbol{s}_{H+1}, \boldsymbol{s}_g) + \lambda_{\text{traj}} C_{\text{traj}}(\boldsymbol{s}_{1:H+1}, \boldsymbol{s}_g) + \lambda_{\text{act}} C_{\text{act}}(\boldsymbol{a}_{1:H}) \tag{5.33}$$

$$C_{\text{goal}}(\boldsymbol{s}_{H+1}, \boldsymbol{s}_g) = \|\tilde{\boldsymbol{p}}(\boldsymbol{s}_{H+1}) - \tilde{\boldsymbol{p}}(\boldsymbol{s}_g)\|_2^2 \tag{5.34}$$

$$C_{\text{traj}}(\boldsymbol{s}_{1:H+1}, \boldsymbol{s}_g) = \frac{1}{H+1} \sum_{t=0}^{H} \|\tilde{\boldsymbol{p}}(\boldsymbol{s}_t) - \tilde{\boldsymbol{p}}(\boldsymbol{s}_g)\|_2^2 \tag{5.35}$$

$$C_{\text{act}}(\boldsymbol{a}_{1:H}) = \frac{1}{H-1} \sum_{t=1}^{H-1} \|\boldsymbol{a}_{t+1} - \boldsymbol{a}_t\|_2^2 \tag{5.36}$$

We test multiple replanning frequencies, ranging from 1 to 32 steps. The optimal weight for each cost component depends primarily on the presence of local minima in the cost landscape. A higher trajectory cost can speed up convergence by providing a denser learning signal, but it may also lead to suboptimal behavior if the agent becomes trapped near obstacles. To use a uniform set of hyperparameters across all replanning frequencies, we set $\lambda_{\text{traj}} = 0.0$ and $\lambda_{\text{act}} = 1 \times 10^{-5}$.

5.5.2 Optimal cost function

In this work, the planning process relies on the Euclidean distance in the learned representation space to guide the agent toward the goal. While this approach succeeds in many scenarios, as demonstrated by our experimental results, it has inherent limitations. The presence of walls and obstacles makes many straight-line paths infeasible, and the true geodesic distance along valid corridors can differ significantly from the direct Euclidean distance between two points. As a result, trajectories that appear low-cost under the Euclidean metric may in reality be impossible to execute, leading to failures in mazes with complex topologies.

Toward Topology-Aware Cost Functions A promising direction is to learn a goal-conditioned value function that captures feasible geodesic distances or implicit notions of reachability. Methods such as Implicit Q-Learning (IQL), introduced in Section 3.2.4.1, can approximate optimal value functions from the same offline trajectory data used to train the world model. The learned Q-function could then replace or augment the Euclidean cost in MPPI, enabling the agent to reason about obstacles and prefer valid paths. We leave this integration as an important direction for future work.

5.6 Results

In this section, we present qualitative and quantitative results to evaluate the performance of our JEPA-based agent in the PointMaze environment. We first analyze the learned visual representations and the effectiveness of the masking mechanism in disentangling static and dynamic features. We then assess the agent's planning capabilities using the Euclidean distance cost, and discuss the observed limitations that motivate future work on topology-aware cost functions.

5.6.1 Static-Dynamic Disentanglement

We begin by evaluating the effectiveness of the masking mechanism in disentangling static and dynamic features in the visual input. The original maze frame, the extracted mask, and the binarized mask are shown in figure 3.

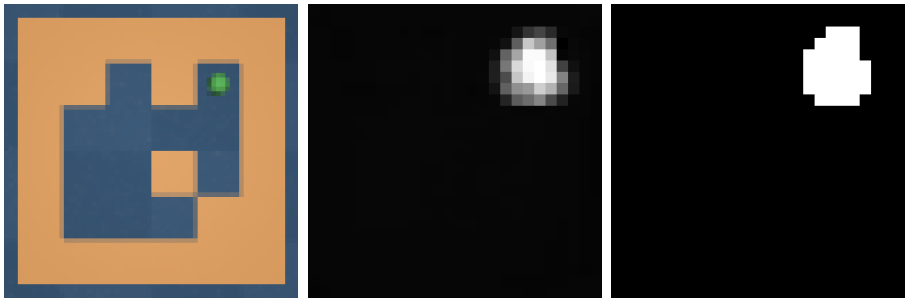


Figure 3: From left to right: original maze frame, extracted mask, and binarized mask.

The mask extractor effectively identifies the parts of the maze that can change from one state to another, which include the agent and the region of reachable states surrounding it. We then visualize the two visual representations, static and dynamic, obtained by applying the mask to the encoded visual input. These are shown in figure 4. The static representation primarily captures the maze layout, walls, and floor texture, while the dynamic representation focuses on the agent and its immediate surroundings. Since we do not apply further processing to the latent representations, some features remain partially entangled; however, the separation between static and dynamic components is still clearly visible.

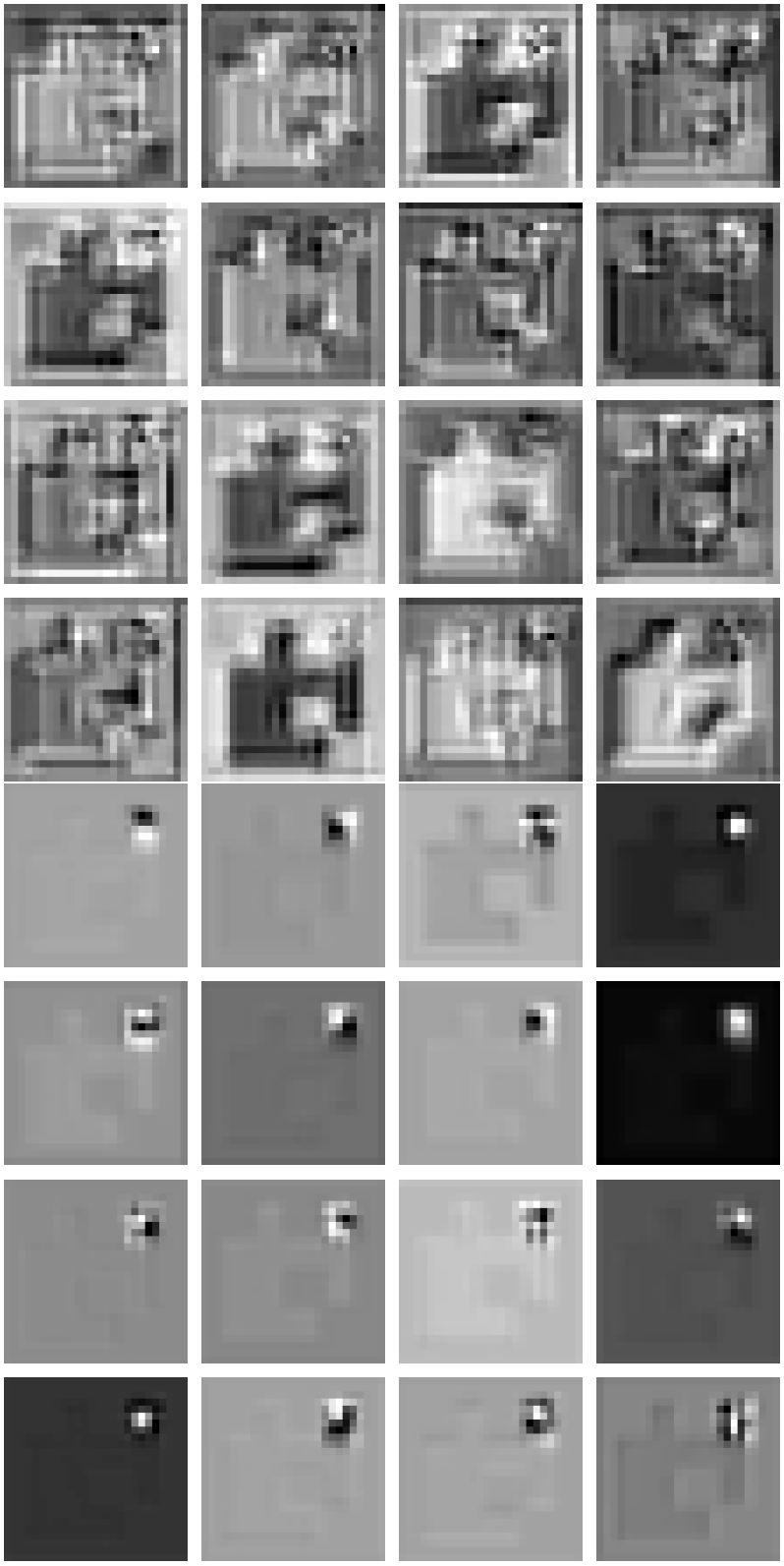


Figure 4: Visual latent representations obtained by applying the mask to the encoded visual input. The upper sixteen channels correspond to the static representation, while the lower sixteen channels correspond to the dynamic representation.

5.6.2 Position Estimation

We then evaluate the accuracy of the position estimates obtained from the mask. In our implementation, the coordinate system is defined in pixel space and projected onto the interval $[-1, 1] \times [-1, 1]$. The figure below shows a heatmap of the Euclidean distance between feasible agent positions and a fixed goal position.

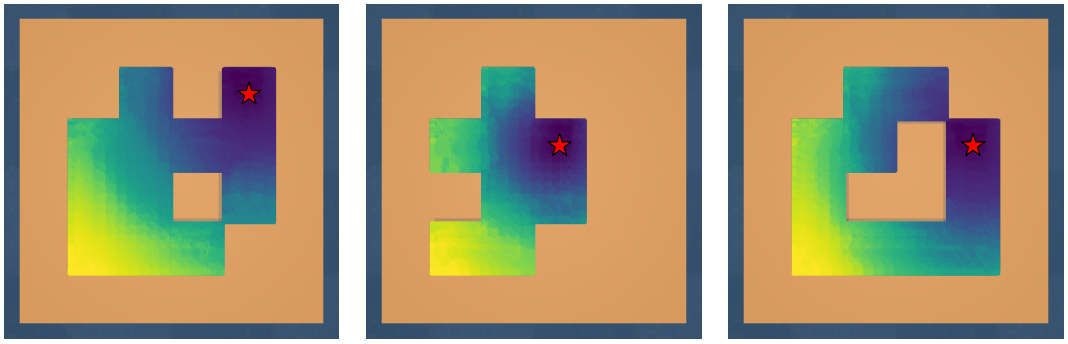


Figure 5: Heatmap of the Euclidean distance between feasible agent positions and the goal position, computed using positions estimated from the mask. The red star marks the goal location.

The distance field increases smoothly as we move away from the goal, as expected from a purely Euclidean metric. However, this measure does *not* account for the maze topology: walls and obstacles are ignored, and two states that are geometrically close but separated by a wall may appear with an artificially small distance.

5.6.3 Planning Performance

We perform 200 evaluations for each replanning frequency, measuring the agent's ability to reach the goal within a tolerance of 2×10^{-1} in Euclidean distance with respect to the ground-truth coordinates. All tests are conducted on unseen maze layouts from the test set, with start and goal positions sampled uniformly from the feasible region of each maze.

The table below summarizes the success rates for different replanning frequencies.

Replanning Frequency	Success Rate (%)
1 step	100.0%
2 steps	100.0%
5 steps	98.5%
10 steps	95.0%
16 steps	91.5%
32 steps	67.0%
∞ steps	26.0%

Table 5.6: Success rates for different replanning frequencies using the Euclidean distance cost.

The success rate decreases as the replanning frequency increases, since the agent has fewer opportunities to correct its trajectory based on updated observations. When no replanning is performed (∞ steps), performance drops sharply due to compounding prediction errors over long horizons.

Some examples of planned trajectories are shown below in figure 6.

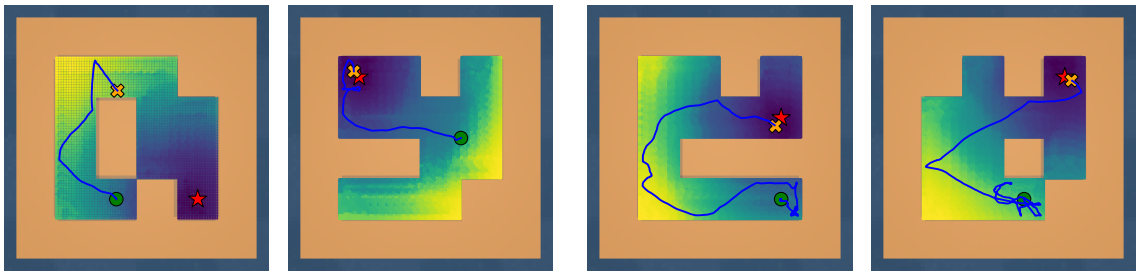


Figure 6: Examples of planned trajectories using the Euclidean distance cost. The first image shows an open-loop trajectory without replanning, which fails to reach the goal even though the initial direction is correct. The other three images show successful trajectories obtained by replanning every 5 steps. The red star denotes the goal location.

Chapter 6

Conclusions

This thesis investigated the application of Joint-Embedding Predictive Architectures to learning world models for goal-conditioned navigation in visually complex environments. By combining self-supervised representation learning with model-based planning, we developed a framework that predicts environment dynamics directly from high-dimensional visual observations without requiring explicit position labels or reward signals during training.

Summary of Contributions We implemented a JEPA-based world model composed of a visual encoder that processes RGB observations and a dynamics predictor that forecasts future latent states. A central contribution is the mask extractor module, which learns to separate static environmental features, walls, floor textures, and background elements, from dynamic components associated with the agent. This decomposition addresses a key challenge in visual world modeling: only a small fraction of each observation is relevant for prediction and planning. By operating in a residual fashion, the predictor focuses exclusively on changes in the dynamic portion, simplifying the learning problem and improving prediction accuracy.

Training JEPA models poses risks of representational collapse, wherein the encoder maps all inputs to a constant representation. Our framework prevents collapse through a combination of techniques: multi-step autoregressive prediction with exponential discounting, inverse dynamics modeling to preserve action information, temporal invariance constraints to enforce static-dynamic separation, and targeted mask regularization. Following the SimSiam methodology, we apply stop-gradient operations on target representations. With carefully tuned loss coefficients, training converges in approximately one hour on a single NVIDIA RTX 4070 Super GPU.

Experimental Validation We evaluated the model in the PointMaze environment, which features second-order dynamics, continuous control, and diverse maze topologies. Training data consisted of 10 000 trajectories across 40 maze layouts, with evaluation on 40 held-out configurations. Qualitative analysis confirms that the mask extractor reliably highlights dynamic regions, primarily the agent and its immediate surroundings,

while suppressing static maze structure, all without explicit supervision.

By thresholding the learned mask and computing its centroid, we extract approximate agent positions in an implicit coordinate system that preserves geometric structure suitable for planning. Using Model Predictive Path Integral control with Euclidean distance cost, our system achieves 100% success rate on unseen mazes when replanning every one or two steps, and maintains 98.5% success with replanning every five steps. These results demonstrate that the learned world model captures sufficient structure for effective goal-conditioned navigation.

Limitations Several limitations of the current approach warrant discussion. First, the position extraction pipeline relies on thresholding and centroid computation, non-differentiable operations that preclude gradient-based trajectory optimization and restrict planning to sampling-based methods such as MPPI. Second, prediction errors accumulate over time, limiting the reliable planning horizon to approximately eight steps without replanning. Third, while Euclidean distance in representation space works well for simple maze topologies, it fails to account for walls and obstacles: states separated by barriers may appear close despite requiring long detours. This limitation is reflected in the declining success rates at lower replanning frequencies, where the planner cannot recover from trajectories that lead toward infeasible paths.

Future Directions Several extensions could address these limitations. Differentiable approximations to mask extraction and centroid computation would enable gradient-based planning methods such as iLQR, potentially improving sample efficiency and solution quality. A hierarchical JEPA architecture could extend the reliable prediction horizon by learning temporal representations at multiple scales. Integrating learned goal-conditioned value functions, such as those produced by Implicit Q-Learning from the same offline data, could provide topology-aware cost signals that respect maze structure. Finally, incorporating probabilistic predictors would support risk-aware planning by quantifying uncertainty in the dynamics model.

Concluding Remarks This thesis demonstrated that Joint-Embedding Predictive Architectures provide an effective framework for learning world models from high-dimensional visual observations without explicit supervision. The learned representations capture meaningful geometric structure and successfully separate static environmental features from the agent's dynamic state, enabling efficient model-based control. The core principles explored here, learning only what is predictable, operating in abstract representation spaces rather than pixel space, and discovering structure through self-supervision, align with emerging views on how intelligent agents can acquire general-purpose world models. By integrating representation learning, world modeling, and planning into a unified framework, this work contributes toward the broader goal of developing autonomous agents capable of learning, adapting, and acting in complex real-world environments.

Bibliography

[1] M. Assran et al. *Self-Supervised Learning from Images with a Joint-Embedding Predictive Architecture*. 2023. arXiv: 2301.08243 [cs.CV]. URL: https://arxiv.org/abs/2301.08243.

[2] M. Assran et al. *V-JEPA 2: Self-Supervised Video Models Enable Understanding, Prediction and Planning*. 2025. arXiv: 2506.09985 [cs.AI]. URL: https://arxiv.org/abs/2506.09985.

[3] A. Baevski et al. "data2vec: A General Framework for Self-supervised Learning in Speech, Vision and Language". In: *CoRR* abs/2202.03555 (2022).

[4] A. Bardes, J. Ponce, and Y. LeCun. "VICReg: Variance-Invariance-Covariance Regularization for Self-Supervised Learning". In: *CoRR* abs/2105.04906 (2021).

[5] A. Bardes et al. *Revisiting Feature Prediction for Learning Visual Representations from Video*. 2024. arXiv: 2404.08471 [cs.CV]. URL: https://arxiv.org/abs/2404.08471.

[6] H. Barlow. "Possible Principles Underlying the Transformations of Sensory Messages". In: *Sensory Communication* 1 (Jan. 1961). DOI: 10.7551/mitpress/9780262518420.003.0013.

[7] X. Chen and K. He. "Exploring Simple Siamese Representation Learning". In: *CoRR* abs/2011.10566 (2020).

[8] J. Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805 (2018).

[9] J. Grill et al. "Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning". In: *CoRR* abs/2006.07733 (2020).

[10] M. Gutmann and A. Hyvärinen. "Noise-contrastive estimation: A new estimation principle for unnormalized statistical models". In: Proceedings of Machine Learning Research 9 (13–15 May 2010). Ed. by Y. W. Teh and M. Titterington, pp. 297–304. URL: https://proceedings.mlr.press/v9/gutmann10a.html.

[11] K. He et al. “Masked Autoencoders Are Scalable Vision Learners”. In: *CoRR* abs/2111.06377 (2021).

[12] K. He et al. “Momentum Contrast for Unsupervised Visual Representation Learning”. In: *CoRR* abs/1911.05722 (2019).

[13] I. Kostrikov, A. Nair, and S. Levine. “Offline Reinforcement Learning with Implicit Q-Learning”. In: *CoRR* abs/2110.06169 (2021).

[14] Y. LeCun. “A Path Towards Autonomous Machine Intelligence Version 0.9.2, 2022-06-27”. In: 2022. URL: https://api.semanticscholar.org/CorpusID:251881108.

[15] L. Li et al. *HiT-JEPA: A Hierarchical Self-supervised Trajectory Embedding Framework for Similarity Computation*. 2025. arXiv: 2507.00028 [cs.LG]. URL: https://arxiv.org/abs/2507.00028.

[16] I. Loshchilov and F. Hutter. “Fixing Weight Decay Regularization in Adam”. In: *CoRR* abs/1711.05101 (2017).

[17] Nguyen and Widrow. “The truck backer-upper: an example of self-learning in neural networks”. In: *International 1989 Joint Conference on Neural Networks*. 1989, 357–363 vol.2. DOI: 10.1109/IJCNN.1989.118723.

[18] M. Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV]. URL: https://arxiv.org/abs/2304.07193.

[19] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *CoRR* abs/1912.01703 (2019).

[20] J. Robinson et al. “Contrastive Learning with Hard Negative Samples”. In: *CoRR* abs/2010.04592 (2020).

[21] R. Shwartz-Ziv et al. *An Information-Theoretic Perspective on Variance-Invariance-Covariance Regularization*. 2024. arXiv: 2303.00633 [cs.IT]. URL: https://arxiv.org/abs/2303.00633.

[22] V. Sobal et al. *Joint Embedding Predictive Architectures Focus on Slow Features*. 2022. arXiv: 2211.10831 [cs.LG]. URL: https://arxiv.org/abs/2211.10831.

[23] V. Sobal et al. *Learning from Reward-Free Offline Data: A Case for Planning with Latent Dynamics Models*. 2025. arXiv: 2502.14819 [cs.LG]. URL: https://arxiv.org/abs/2502.14819.

[24] Y. Tian, X. Chen, and S. Ganguli. “Understanding self-supervised Learning Dynamics without Contrastive Pairs”. In: *CoRR* abs/2102.06810 (2021).

[25] M. Towers et al. *Gymnasium: A Standard Interface for Reinforcement Learning Environments*. 2025. arXiv: 2407.17032 [cs.LG]. URL: https://arxiv.org/abs/2407.17032.

[26] T. Wang and P. Isola. "Understanding Contrastive Representation Learning through Alignment and Uniformity on the Hypersphere". In: *CoRR* abs/2005.10242 (2020).

[27] J. Zbontar et al. "Barlow Twins: Self-Supervised Learning via Redundancy Reduction". In: *CoRR* abs/2103.03230 (2021).

[28] G. Zhou et al. *DINO-WM: World Models on Pre-trained Visual Features enable Zero-shot Planning*. 2025. arXiv: 2411.04983 [cs.RO]. URL: https://arxiv.org/abs/2411.04983.