Advances in Deep Learning: Multivariate Time Series Forecasting to Single- and Multi-Agent Systems

by

Azad Deihim

A thesis submitted in partial fulfillment of the requirements
for the degree of *Doctor of Philosophy*

City St George's, University of London

Department of Computer Science

Supervisors:
Dr. Eduardo Alonso
Dr. Dimitra Apostolopoulou

August 2025

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisors, Dr. Dimitra Apostolopoulou and Dr. Eduardo Alonso, for their continuous guidance, support, and encouragement throughout the course of this PhD. Their expertise and insights have been invaluable to both my research and my personal development as a researcher.

I am also grateful to my friends, colleagues, and collaborators within the School of Science and Technology, across the Department of Engineering and the Department of Computer Science that I have had the pleasure of meeting during my PhD.

Azad Deihim
London, August 2025

Contents

List of Figures

List of Tables

List of Acronyms

Acronym	Definition
ACOPF	Alternating Current Optimal Power Flow
DCOPF	Direct Current Optimal Power Flow
OPF	Optimal Power Flow
IPS	Interior Point Solver
GNN	Graph Neural Network
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
MTS	Multivariate Time Series
RL	Reinforcement Learning
MARL	Multi-Agent Reinforcement Learning
VAE	Variational Autoencoder
VQ-VAE	Vector Quantized Variational Autoencoder
EMA	Exponential Moving Average
STTRE	Spatio-Temporal Transformer with Relative Embeddings
MATWM	Multi-Agent Transformer World Model
SMAC	StarCraft Multi-Agent Challenge
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
KL	Kullback–Leibler Divergence
IEEE	Institute of Electrical and Electronics Engineers

Chapter 1

Introduction

Usage of machine learning technology is becoming increasingly embedded in modern economic activity. According to the UK Office for National Statistics, nearly a quarter (23%) of UK businesses reported using some form of artificial intelligence technology in late 2025, up from 9% in 2023 [Office for National Statistics, 2025]. This rapid growth reflects the expanding role of machine learning systems in forecasting, optimization, automation, and decision-making across various industries. While machine learning technologies can be deployed across a wide range of sectors, examples of current use can be found in finance, pharmaceuticals and biotechnology, energy, manufacturing, transport, and healthcare. Within these sectors, it can be used for tasks such as forecasting demand and prices [Phuoc et al., 2024], detecting anomalies and faults [Moharam et al., 2025], enabling real-time control of large-scale systems [Zhou et al., 2020c], and supporting scientific discovery [Dara et al., 2022]. This has created a growing need for sophisticated learning-based methods capable of addressing these complex real-world problems.

The last decade has seen rapid advances in machine learning, particularly in deep learning. Machine learning is a subfield of artificial intelligence that intersects computer science and statistics [Goodfellow et al., 2016]. It uses data-driven statistics-based algorithms to build models that can learn from patterns in data, enabling systems to generate insights, make predictions, or automate decision-making without being explicitly programmed or hard-coded. Deep learning is a specialized branch of machine learning that focuses on training artificial neural networks (NN) with multiple layers (deep architectures) to learn complex representations of data. Given raw data, NNs can extract increasingly abstract learned features. These deep learning models have achieved state-of-the-art performance in complex areas such as computer vision [Trigka and Dritsas, 2025], natural language processing [Treviso et al., 2023], and reinforcement learning [Han et al., 2023a] where classical machine learning has otherwise not been nearly as effective.

Classical machine learning methods, such as support vector machines [Cortes and Vapnik, 1995], decision trees and random forests [Breiman, 2001], and statistical models like ARIMA or VAR [Box et al., 2015], have been widely used in prediction and control tasks for decades. These methods perform well in relatively low-dimensional settings where the data distribution is stable and relationships are simple or approximately linear [Hastie et al., 2009]. However, they exhibit several key limitations that make them unsuitable for the complex, high-dimensional, and highly structured problems addressed in this thesis.

First, classical methods rely heavily on manual feature engineering. Designing informative features requires significant domain expertise and often must be repeated for each new application. This makes such methods time-consuming to deploy and difficult to generalize across domains. In contrast, modern deep learning architectures can learn hierarchical feature representations directly from raw data [Dee, 2015, Goodfellow et al., 2016], reducing dependence on hand-crafted inputs. Second, most classical models assume linear or shallow nonlinear relationships between inputs and outputs. While kernel methods like support vector machines can capture some nonlinearities, they scale poorly with data size and dimensionality [Vapnik, 1995], quickly becoming computationally expensive. Statistical forecasting models such as ARIMA and VAR similarly struggle with scalability, as they model only short-term dependencies and assume stationarity [Box et al., 2015]. This limits their ability to capture long-range temporal patterns and cross-variable dependencies in sequences or multivariate data [Lara-Benítez et al., 2021]. Third, classical approaches are limited in their ability to model complex relational structures.

In many modern applications, such as power systems optimization, multivariate time series forecasting, or multi-agent coordination, the underlying data is structured as graphs or sequences with rich dependencies [Vaswani et al., 2017]. Treating these elements as independent or only weakly correlated, as classical methods often do, leads to suboptimal performance. Finally, while classical models are typically more interpretable and computationally lightweight, they fall short when accuracy is critical. They lack the expressiveness required to handle nonlinear, high-dimensional environments and often fail to generalize effectively beyond their training conditions. These limitations motivate the use of deep learning architectures, which can automatically extract meaningful features, capture complex dependencies, and adapt more readily to large-scale and dynamic real-world systems [Vaswani et al., 2017, Mnih et al., 2015, Hernandez-Leal et al., 2019].

Despite the drawbacks of classical machine learning, deep learning is not a perfect solution; it faces several well-recognized challenges that hinder its global applicability. First, neural networks often require significant amounts of labeled data, which may not always be available for a given domain specific problem [Dee, 2015]. Second, they generally demand more computational resources than classical machine learning models, restricting usage in resource-constrained applications [Shorten and Khoshgoftaar, 2019]. Third, tuning and optimization of hyperparameters can be finicky, often times small changes in hyperparameters can make or break performance [Ilemobayo et al., 2024]. Fourth, while deep learning models excel at capturing complex patterns in data, this same flexibility can cause them to fit spurious patterns or noise in the training set, a phenomenon known as overfitting [Goodfellow et al., 2016]. Fortunately, many of these challenges can be mitigated, or even entirely avoided, thanks to the substantial advancements and methodological improvements that have already emerged in the field of deep learning. But given the depth and complexity of the field, there is still much more to uncover and many more improvements to be made.

Advancements in deep learning subfields such as computer vision, natural language processing, and reinforcement learning have enabled significant progress across many real-world domains. Examples include medical imaging and diagnosis in healthcare, risk forecasting in finance, perception and navigation for autonomous vehicles, intrusion detection in cybersecurity, load forecasting and grid optimization in energy systems, motion planning in robotics, and adaptive tutoring in education [Trigka and Dritsas, 2025, Treviso et al., 2023, Han et al., 2023a, Aslam et al., 2025, Mienye et al., 2024, Rane et al., 2024, Kuutti et al., 2019]. Despite the progress deep learning has brought to these domains, deep learning applications to many other real-world problems remain untapped or underexplored because the deep learning area that coincides with these specific problems are not as well studied. Numerous subfields within deep learning are still under-researched, even though they hold substantial potential for widespread, practical application.

My doctoral research began with the application of deep learning techniques to problems in electrical power systems. Over time, the focus shifted from the application domain toward the underlying deep learning methodologies themselves, as it became evident that the specific subfields most relevant to solving these power system challenges—namely, multivariate time series modeling and multi-agent reinforcement learning —remain underexplored and lag behind their more mature single-variable and single-agent counterparts [Lara-Benítez et al., 2021, Lim and Zohren, 2021, Hernandez-Leal et al., 2019, Papoudakis et al., 2021]. Existing methods in these areas often lack the complexity and

expressiveness needed to address the intricacies of real-world power systems in a substantial way. Consequently, while this thesis includes work on deep learning applications to electrical power systems, particularly the optimal power flow problem, its primary emphasis is on advancing these underdeveloped yet critically important areas of deep learning: multivariate time series and multi-agent reinforcement learning.

Research Contributions

This thesis makes three principal contributions in power system optimization, time series forecasting, and multi-agent learning, each with further sub-contributions detailed in their respective chapters.

1. A graph neural network and reinforcement learning framework that learns warm-starts for nonlinear AC optimal power flow solvers, reducing convergence time while preserving solver accuracy.
2. A novel Transformer architecture (STTRE) that explicitly models spatio-temporal dependencies, achieving substantial accuracy improvements on benchmark multivariate time series datasets.
3. A Transformer world model (MATWM) that improves sample efficiency and coordination in cooperative multi-agent environments.

These contributions are discussed in detail below.

1. Learning Warm-Starts for AC Optimal Power Flow via Graph Neural Networks and Reinforcement Learning. I address the problem of solving nonconvex AC Optimal Power Flow (ACOPF) problems in a wide range of electrical power system configurations. Optimal power flow (OPF) is a crucial task in power system management and control; accurate and time-efficient solutions for OPF are necessary to ensure cost-efficient and reliable power system operation [Wood et al., 2013]. A common approach to simplifying OPF is to apply a DC approximation, which linearizes the power flow equations and makes the problem convex and thus more tractable [Sun and Tesfatsion, 2006]. While computationally efficient, this approximation introduces significant errors because it neglects reactive power flows, voltage magnitudes, and line losses. As a result, the DC-OPF solution often deviates substantially from the true AC-OPF optimum, and in many cases, the DC optimal solution is not even feasible when evaluated under the full AC constraints. On the other hand, solving the full ACOPF problem is computationally expensive and scales poorly with system size, limiting its applicability for real-time operation. Classical numerical methods, such as the Newton–Raphson algorithm and interior-point solvers, can in principle achieve highly accurate solutions, but they often require significant computational time to converge, especially for large-scale or heavily congested networks. In response, a growing body of research has explored machine learning and deep learning approaches as alternatives. While these methods offer substantial speedups by learning direct mappings from inputs to approximate solutions, they generally fall short of exact optimality due to the inherent trade-offs in learning-based

generalization and their limited ability to guarantee feasibility under full AC constraints. Rather than designing an entirely new solver, I propose a hybrid approach for solving ACOPF, a nonlinear and nonconvex optimization problem, by combining the speed of a trained deep learning model with the accuracy of existing numerical solvers. The proposed framework uses a graph neural network (GNN) to exploit the graph structure of a power system in conjunction with proximal policy optimization, a deep reinforcement learning algorithm, to compute initial guesses for an interior point solver (IPS), providing a warm start, allowing the solver to converge in fewer iterations. Existing literature that explores warm start ACOPF solutions using machine learning choose to compute initial guesses that are trained to be feasible and cost-minimizing. This approach trains the GNN-based reinforcement learning agent to produce an output that minimizes IPS convergence time by designing a reward function that is a function of the IPS convergence time. The proposed framework is evaluated using IEEE test case environments, using PyPower's IPS-based ACOPF solver and a GNN-based framework that computes ACOPF solutions directly as baselines, demonstrating significantly improved convergence times.

This work has been published in *Electric Power Systems Research (EPSR), 2024* [Deihim et al., 2024].

- DOI: `https://doi.org/10.1016/j.epsr.2024.110782`
- Code: `https://github.com/AzadDeihim/ACOPF-PPO-GNN`

2. STTRE: A Spatio-Temporal Transformer for Multivariate Time Series Forecasting. A time series is a sequence of data points collected or recorded at successive, usually equally spaced, points in time. In a multivariate time series (MTS), each observation at time t consists of multiple interdependent variables. Time series data are prevalent in many real-world domains, including energy systems (e.g., electricity demand, renewable generation), finance (e.g., stock prices, trading volumes), transportation (e.g., traffic flow, ride-sharing demand), and healthcare (e.g., patient vital signs). The primary objective in time series analysis is to extract meaningful patterns from past observations to understand the underlying processes and to forecast future values. This work originally began as research for Locational Marginal Pricing (LMP) forecasting, a method used in electricity markets to determine the price of electricity at specific locations within the power grid, reflecting the cost of supplying an additional increment of power at that particular point, but was later expanded . At its core, LMP forecasting is primarily a multivariate time series forecasting problem. Unlike static data, time series observations exhibit temporal dependencies over periods of time, such as trends or seasonal effects. But multivariate time series observations also have correlations and dependencies across variables [Lara-Benítez et al., 2021, Lim and Zohren, 2021], which must be captured for accurate modeling and prediction. Upon doing an extensive literature review in the space of multivariate time series analysis, I found that much of the research often neglects the spatial aspects of the data. Additionally, I found that the use of Transformer-based architectures, a state-of-the-art architecture for sequence learning tasks, was not well explored.

Drawing inspiration from a popular natural language processing model, the Transformer, and I propose the Spatio-Temporal Transformer with Relative Embeddings (STTRE) to

address multivariate time series forecasting. This work primarily focuses on developing a Transformer-based framework that can fully exploit the spatio-temporal nature of a multivariate time series by incorporating several of the Transformer's key components, but with augmentations that allow them to excel in multivariate time series forecasting. Current Transformer-based models for multivariate time series often neglect the data's spatial component(s) and utilize absolute position embeddings as their only means to detect the data's temporal component(s), which I show is flawed for time series applications. The lack of emphasis on fully exploiting the spatio-temporality of the data can incur subpar results in terms of accuracy. I redesign relative position representations, which I rename to relative embeddings, to unveil a new method for detecting latent spatial, temporal, and spatio-temporal dependencies more effectively than previous Transformer-based models. These relative embeddings are then coupled with a restructuring of the Transformer's primary sequence learning mechanism, multi-head attention, in a way that allows for full utilization of relative embeddings, thus achieving up to a 24% improvement in accuracy over other state-of-the-art multivariate time series models on a comprehensive selection of publicly available multivariate time series forecasting datasets.

This work has been published in *Neural Networks, 2023* [Deihim et al., 2023].

- DOI: `https://doi.org/10.1016/j.neunet.2023.09.039`
- Code: `https://github.com/AzadDeihim/STTRE`

3. MATWM: A Transformer World Model for Sample-Efficient Multi-Agent Reinforcement Learning.

Reinforcement learning is a framework for optimizing decision-making processes through trial-and-error interaction, where an agent learns by observing the consequences of its actions and receiving reward signals that quantify their quality. In many practical scenarios, reinforcement learning involves multiple agents that share the same environment and must either cooperate or compete to achieve their goals. This multi-agent setting introduces additional challenges, including coordination, non-stationarity, partial observability, and scalability. A key limitation of reinforcement learning, particularly in multi-agent settings, is data inefficiency: agents often require millions of interactions to learn effective policies, making training costly or impractical in many real-world applications. This is especially problematic in physical systems such as AC optimal power flow, where each interaction may depend on computationally expensive simulation which becomes more expensive as system scale increases. As a result, many high-impact physical domains remain inaccessible to conventional reinforcement learning methods. Improving sample efficiency can therefore play a key enabling role in making reinforcement learning practical for real-world physical systems.

World models seek to alleviate data inefficiency in reinforcement learning by learning approximations of the environment and its dynamics, allowing agents to generate additional training experience more efficiently [Ding et al., 2024]. As a result, improving sample efficiency can play an important role in making reinforcement learning practical for physical domains. Using the world model, agents can "imagine" new data to learn from, significantly lessening the amount of interactions required from the real environment to learn a

task. Likely due to the added complexities of modeling multi-agent environments, the use of world models to improve sample efficiency in reinforcement learning has been widely explored in single-agent settings relative to multi-agent settings where the research lags.

A key limitation of reinforcement learning, particularly in multi-agent settings, is data inefficiency: agents often require millions of interactions to learn effective policies. This is especially problematic in physical systems, where each interaction may require computationally expensive simulation. World models seek to alleviate this issue by learning approximations of the environment dynamics, allowing agents to generate imagined experience and reduce reliance on real environment interactions. Although world models have been widely explored in single-agent reinforcement learning, their use in multi-agent settings remains comparatively less developed due to the additional complexity of modeling interactions between agents.

I present the Multi-Agent Transformer World Model (MATWM), a novel transformer-based world model designed for multi-agent reinforcement learning in both vector- and image-based environments. MATWM combines an independent imagination framework with a semi-centralized critic and a teammate prediction module, enabling agents to model and anticipate the behavior of others under partial observability. To address non-stationarity, I incorporate a prioritized replay mechanism that trains the world model on recent experiences, allowing it to adapt to agents' evolving policies. MATWM is evaluated on a broad suite of benchmarks. MATWM achieves state-of-the-art performance, outperforming both model-free and prior world model approaches, while demonstrating strong sample efficiency, achieving near-optimal performance in as few as 50K environment interactions. Ablation studies confirm the impact of each component, with substantial gains in coordination-heavy tasks. This work is available as a preprint on *arXiv, 2025* [Deihim et al., 2025].

- arXiv: `https://arxiv.org/abs/2506.18537`
- Code: `https://github.com/AzadDeihim/MATWM`

The thesis is structured as follows: Chapter 2 provides background on deep learning methods in optimization, forecasting, and reinforcement learning, with an emphasis on structured and relational architectures. Chapter 3 presents a GNN-based warm-starting method for ACOPF. Chapter 4 introduces the STTRE model for MTS forecasting. Chapter 5 describes the MATWM architecture for multi-agent learning. Chapter 6 concludes the thesis and outlines potential future directions, including applications to real-world systems.

Chapter 2

Background and Literature Review

2.1 Neural Networks

Artificial neural networks, referred to hereafter as neural networks, consist of computational units called nodes or neurons, arranged in a three-layered structure: the input layer, hidden layer(s), and output layer. A standard feed-forward neural network contains one input layer and one output layer, with one or more hidden layers between them — which will determine the depth of a neural network. Deeper networks can learn more complex, or higher order relationships [Goodfellow et al., 2016, LeCun et al., 2015]. Figure 2.1 shows a graphical representation of a neural network, highlighting the structure of the layers and their relationship with each other.

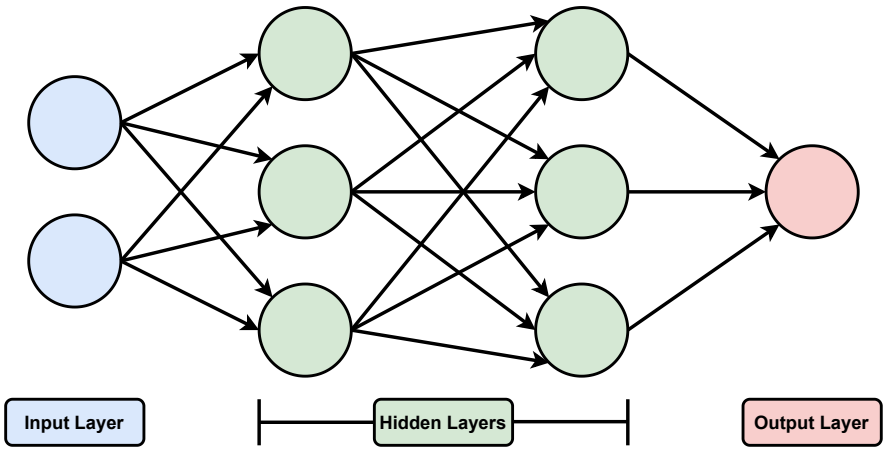

Figure 2.1: The input layer receives the data, with the number of nodes equal to the dimensionality of the input features. The output layer produces the final prediction, often through a single node, though multiple nodes may be used for multi-class or multi-output tasks. Between them lie one or more hidden layers, whose size is determined by the user. Each layer consists of nodes fully connected to those of the subsequent layer, with learnable parameters in the form of a weight matrix and a bias vector.

The traversal from any layer to the next is computed via linear transformation followed by an activation function:

$$X_{i+1} = Activation(X_i W_{(i,i+1)} + b_{(i,i+1)}), \tag{2.1}$$

where X_i denotes the vector of nodes at layer i, $W_{(i,i+1)}$ denotes the weight matrix between layers i and $i+1$, and $b_{(i,i+1)}$ denotes the bias vector between layers i and $i+1$. Note that for the transition from the input layer to the first hidden layer, X will be the original input, flattened into a vector. Each W and b is learned and optimized using back propagation; the process of optimizing the weights and biases by minimizing an error rate calculated from a predefined loss function. Activation functions introduce nonlinearity into neural networks, allowing them to approximate complex functions beyond simple linear transformations.

Modern neural network architectures rarely consist of only input, hidden, and output layers. Neural network layers, also commonly referred to as linear layers or feed-forward layers, are typically placed carefully throughout the architecture to supplement other learning mechanisms, and do not always use activation functions. Additionally, there are several other commonly used configurations and operations that leverage learned weights, such as convolution or recurrence. Equation 2.1 represents just one of many possible formulations, albeit the most widely used.

2.1.1 Backpropagation

The backpropagation algorithm [Rumelhart et al., 1986b] is vital to neural network training. It computes the gradient of the loss function with respect to the weights of the network by recursively applying the chain rule of calculus from the output layer backwards through all hidden layers. This efficient calculation enables gradient-based optimization methods to update millions of parameters, making the training of deep neural networks feasible. Backpropagation minimizes a loss function $L(\theta)$ by computing $\nabla_\theta L$ through successive derivatives of each layer's activations and weights.

2.1.2 Activation Functions

Activation functions provide the non-linear transformations that allow neural networks to approximate complex functions. Without such nonlinearities, even deep architectures would effectively collapse into a single linear model.

The sigmoid function is one of the earliest choices, defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}},$$

which smoothly maps real-valued inputs to the interval $(0, 1)$ and enables probabilistic interpretations in binary classification tasks. However, the sigmoid saturates for large positive or negative inputs, leading to vanishing gradients that hinder training in deep networks [Rumelhart et al., 1986b].

The hyperbolic tangent (tanh) activation,

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

addresses the non-centered output of the sigmoid by mapping values to $(-1, 1)$. This often improves convergence, though tanh still suffers from vanishing gradients for extreme input values [Rumelhart et al., 1986b].

The rectified linear unit (ReLU) is defined as:

$$\mathrm{ReLU}(x) = \max(0, x).$$

Compared to sigmoid or hyperbolic tangent functions, ReLU greatly reduces the vanishing gradient problem, is computationally efficient, and often accelerates training convergence [Nair and Hinton, 2010, Glorot et al., 2011].

2.1.3 Optimization Algorithms

To minimize the loss function, various optimization methods are employed:

- **Stochastic Gradient Descent (SGD)** updates parameters based on noisy gradient estimates from mini-batches, which improves scalability to large datasets [Robbins and Monro, 1951].
- **Momentum** accelerates SGD by accumulating a moving average of past gradients, smoothing the update direction [Polyak, 1964].
- **Adam** combines the advantages of RMSProp and Momentum by maintaining per-parameter adaptive learning rates and exponentially decaying averages of past gradients [Kingma and Ba, 2015]. Adam is widely used as the default optimizer in deep learning due to its stability and fast convergence.

2.2 Graph Neural Networks

Given a graph with a set of nodes or vertices V and a set of edges that denote connections between nodes E, GNNs use optimizable operations, such as aggregation [Hamilton et al., 2017], pooling [Ma et al., 2020], attention [Veličković et al., 2018], or convolution [Kipf and Welling, 2016], that consider entities of a graph as interconnected rather than independent, taking into account their relationships with one another. The identity of a node or edge can be represented by one or more values; for example, in the context of power systems, nodes can denote buses and be represented by values such as real power, reactive power, voltage, etc. In contrast, edges can represent the lines that connect two buses and be represented by values such as resistance, conductance, susceptance, current, etc. Graph-structured data exists in various domains, from street traffic to molecular biology to social networks. Prediction tasks for graph-structured data can fall into three categories: graph-level, node-level, or edge-level. Graph-level tasks involve computing all properties of an entire graph; node-level tasks are concerned with predicting values that represent the identity of a node; and edge-level tasks can involve predicting where edges in a graph should be or determining the values that represent that edge's identity.

Graph convolutional networks, a popular type of GNN, utilize an operation known as graph convolution, which is similar to the convolution operation used in a convolutional neural network for images; the new representation of a node in a graph becomes an amalgamation of itself and its neighbors. The model computes a transformation of each node using feature information from all its neighbors and itself, allowing the network to consider nodes as interconnected rather than independent entities. The convolution operation translates well between images and graphs simply because images are graphs,

too, where each pixel is nodally connected to adjacent pixels. While the idea of graph convolution is similar to the convolution operation used in a convolutional neural network for images, their mathematical formulations differ; also, graphs require an order-invariant operation, whereas the convolution operation used for images is not order-invariant. In this thesis, I adopt the graph convolution formulation in [Morris et al., 2019], shown below:

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \mathbf{W}_2 \sum_{j \in \mathcal{N}(i)} e_{j,i} \cdot \mathbf{x}_j, \tag{2.2}$$

where $x_{(\cdot)}$ is the value(s) of a node, $e \in \mathbb{R}^{||\mathcal{N}|| \times ||\mathcal{N}||}$ is the adjacency matrix, where $||(\cdot)||$ denotes cardinaility of the set, and $W_{(\cdot)}$ are learned weight matrices.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models particularly well-suited for processing data with a grid-like structure, such as images. CNNs are built upon the convolution operation, which applies a learnable filter to local regions of the input space to extract spatial features. Unlike traditional fully connected neural networks, CNNs take advantage of spatial locality and parameter sharing, significantly reducing the number of learnable parameters and improving scalability to higher-dimensional inputs [LeCun et al., 1989]. CNNs have shown strong performance in a wide variety of domains, including computer vision, signal processing, and time series forecasting.

Each layer of a CNN applies a set of convolutional filters across the input, followed by a nonlinear activation function such as ReLU. Pooling operations are often interleaved between convolutional layers to downsample feature maps, reducing dimensionality while preserving salient features [LeCun et al., 1989]. In typical CNN architectures, early layers capture low-level features like edges or textures, while deeper layers extract higher-level semantic representations. These representations are then passed through one or more fully connected layers to perform classification, regression, or other downstream tasks.

While CNNs were originally developed for image data, their core operations—localized filtering and weight sharing—are broadly applicable to any structured input. For instance, time-series or tabular data can be embedded into pseudo-image formats to enable convolutional feature extraction.

The convolution operation for a two-dimensional input $X \in \mathbb{R}^{H \times W}$ using a filter $W \in \mathbb{R}^{k_h \times k_w}$ and bias term b is given by:

$$Y_{i,j} = \sigma \left(\sum_{m=0}^{k_h - 1} \sum_{n=0}^{k_w - 1} W_{m,n} \cdot X_{i+m,j+n} + b \right), \tag{2.3}$$

where $Y_{i,j}$ is the output at position (i, j), k_h and k_w are the filter's height and width,

respectively, and $\sigma(\cdot)$ is a nonlinear activation function (e.g., ReLU). This operation enables the network to learn spatial features from localized patches of the input.

Although CNNs can model local dependencies effectively, they do not inherently model long-range relationships unless many layers are stacked, which can increase model complexity and training time [Bai et al., 2018]. Additionally, CNNs require fixed input sizes and may not generalize as naturally as graph-based models in domains with variable topology. For this reason, CNNs are best employed in applications where data structure is regular and relationships among variables are spatially or temporally localized. In this study, CNNs serve as a point of comparison for graph-based methods, which are more appropriate for explicitly structured systems like power grids.

2.4 Transformer

The Transformer [Vaswani et al., 2017] is an attention-based neural network developed for natural language translation. Attention mechanisms allow neural networks to weight different parts of an input according to their relevance. The use of attention in neural networks across several areas of machine learning has been widely studied ([Chaudhari et al., 2021]). While the Transformer was initially developed for natural language translation, many have studied its application outside of natural language processing (NLP), discovering merit across many sequence learning tasks and beyond sequence learning ([Han et al., 2023b]). Many of the modules and mechanisms utilized in the Transformer were designed solely for sequence-to-sequence processing of words and are incompatible with other tasks. As a result, architectures tend to differ significantly from that of the original Transformer in non-NLP applications.

The Transformer uses an encoder to process an input sequence to create a new encoded representation of the input, which a decoder can process and translate into a desired output sequence. The primary learning mechanism in the transformer decoder, masked multi-head attention, requires the output to be a sequence. For multivariate time series forecasting, this work predicts one value rather than a sequence; thus, using a decoder is neither necessary nor applicable in this work, so this work will focus on the encoder.

The Transformer encoder comprises the following components: multi-head attention, feed-forward, and normalization layers. Figure 2.2 shows a diagram of the encoder.

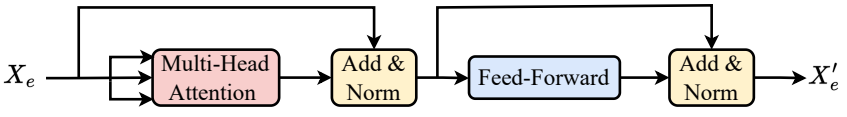

Figure 2.2: Diagram of the Transformer encoder. The encoder, which processes X_e to compute X'_e, comprises multi-head attention and a feed-forward layer, with normalization after each.

Firstly, the raw input X is embedded. In the multivariate time series case, $X \in \mathbb{R}^{l \times m}$ is cast to $X_e \in \mathbb{R}^{l \times m \times d}$, where l is the sequence length, m is the number of variables, and d is the size of the embedding dimension (in the univariate case, m is 1). Next, multi-head

attention processes X_e to produce a new matrix $z \in \mathbb{R}^{l \times m \times d}$, a matrix of attention scores. A skip connection is used to sum z with X_e. The resultant matrix is then normalized, producing $z_n \in \mathbb{R}^{l \times m \times d}$. A two-layer feed-forward followed by an activation function is used to transform z_n into a new matrix with the same dimension; the elected activation function will be LeakyReLU:

$$\mathrm{FF}(z_n) = \mathrm{LeakyReLU}(W_1 z_n + b_1)W_2 + b_2, \tag{2.4}$$

$$\mathrm{LeakyReLU}(x) = \left\{ \begin{array}{ll} x, & \text{if } x \geq 0 \\ 0.01x, & \text{if } x < 0 \end{array} \right\}, \tag{2.5}$$

where $W_{(.)}$ and $b_{(.)}$ are weight matrices and bias vectors respectively. The feed-forward output is summed with the input, z_n. The sum is then normalized, producing the final output of the encoder, $X'_e \in \mathbb{R}^{l \times m \times d}$, highlighting important features and dependencies within X_e.

The encoder also utilizes a layering mechanism, where the encoder is duplicated k times: the input of the i'th encoder sublayer will be the output of encoder sublayer i-1, and the output of the k'th layer will be the final output of the encoder. Note that learned weights are not shared across sublayers.

As a standalone module, the encoder cannot generate a prediction; it simply creates an encoded data representation. With the removal of the decoder, an alternative learning mechanism is required to translate the encoded representation into a prediction.

2.4.1 Embeddings

The encoder operates on an embedded sequence; the embedding layer casts each sequence's elements into a higher-dimensional space, where each dimension can represent a different characteristic, feature, or contextual meaning regarding its respective element. Elements closer to each other in this multidimensional space are more likely to be closely related. Element embeddings are more intuitive when operating on words, as it is widely recognized that words can have different meanings depending on their context, but this is also true for elements in a time series, as specific values may have different contextual meanings based on their surrounding values. These element embeddings are learned during the model's training via a set of learned weights. In the original Transformer, which had a discrete input space, an embedding vector for each unique word is stored in a dictionary. In the continuous case, where an infinitely large dictionary is infeasible, a learned weight matrix $W_e \in \mathbb{R}^{l \times dl}$ and bias vector $b_e \in \mathbb{R}^l$ are used to linearly transform the sequence via $W_e X + b_e$, thus casting it into the embedding dimension. Figure 2.3 provides a visualization of the element embedding process.

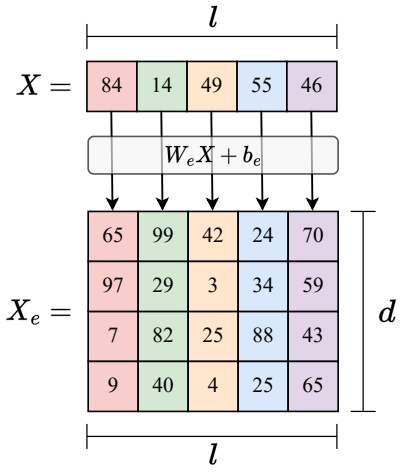

Figure 2.3: Visualization of the element embedding process on an l length sequence.

Another type of embedding, position embedding, is used to implant position information into each element. Since multi-head attention has no built-in mechanism to understand the order of a sequence, absolute position embeddings are necessary to understand temporal information. Each position t_i in $T = \{t_1, t_2, t_3, \cdots, t_l\}$ will be translated into a higher dimensional vector, resulting in the absolute position embedding matrix $PE \in \mathbb{R}^{l \times d}$, where each position t_i will be represented by a vector of d different values. The Transformer introduced sinusoidal position embeddings of varying frequencies, taking the form:

$$PE(t_i, j) = \sin(\frac{t_i}{10000^{2j/d}}) \qquad \forall_t \in T, \tag{2.6}$$

$$PE(t_i, j) = \cos(\frac{t_i}{10000^{(2j+1)/d}}) \qquad \forall_t \in T, \tag{2.7}$$

where $j = \{1, 2, 3, ..., \frac{2}{d}\}$. Sinusoidal embeddings work well for sequence learning tasks with sequences of varying lengths, as they can generalize to sequence lengths that are unseen during training. Their flaw is that they may overpower or be overpowered by element embeddings when summed together, as orthogonality is not guaranteed to remain intact after they are added. Some have experimented with concatenating element embeddings and positional embeddings rather than summing them to ensure that they occupy orthogonal spaces ([Huang et al., 2018]), but this is an impractical approach in many applications as it increases memory requirements significantly. Time2Vec ([Kazemi et al., 2019]) experiments with learned sinusoidal position representations to alleviate orthogonality issues. Since then, many tried fully learnable absolute position embeddings, where an embedding for every possible position is stored in a dictionary ([Devlin et al., 2018]). A benefit of fully learnable absolute position embeddings is that they can learn to occupy a virtually orthogonal space in relation to the element embeddings, mimicking the effect of concatenation without the added memory constraints. However, fully learnable absolute position embeddings do not work well in tasks with varying sequence lengths, as they cannot generalize to sequence lengths unseen during training — the learned dictionary will not contain entries for positions beyond the maximum position seen during training. Also, higher positions seen infrequently during training may have undertrained

weights. In time series tasks, where there is more control over the length of sequences, varying sequence lengths will not be a complication.

For use in multivariate time series, ([Grigsby et al., 2021]) experimented with variable embeddings, utilizing a similar method to fully learnable absolute position embeddings to embed variable information into each element, thus informing multi-head attention about variable information regarding each element in the sequence.

In 2018, relative embeddings were created as a novel way to represent positional information for multi-head attention by using an embedding matrix of relative distances between all sequence element pairs, granting minor accuracy improvement over absolute position embeddings ([Shaw et al., 2018]). Note that relative embeddings are more representative of the relationship between two elements than an actual numerical distance between their positions. They function by calculating a matrix of pairwise embeddings between any two elements in the input sequence, informing attention about the relationship between two positions. The memory requirements associated with relative embeddings made them infeasible for long sequence lengths, but they were later optimized to reduce memory requirements significantly ([Huang et al., 2018]).

Each type of embedding offers a unique benefit when computing a new representation of the input data. This thesis will utilize many of the aforementioned embedding methods.

2.4.2 Multi-head Attention

At the core of the Transformer encoder is multi-head attention, a type of self-attention mechanism that computes an attention function several times in parallel, concatenating each computation. Self-attention describes a type of attention mechanism that can relate different positions of a sequence and compute a new representation of that sequence; this new representation highlights parts of the sequence and its embedding space that warrant more attention.

Multi-head attention is divided into h heads, each attending to $\frac{1}{h}$ of the input space. The output is $z \in \mathbb{R}^{l \times d}$, or in the multivariate case, $z \in \mathbb{R}^{l \times m \times d}$, a matrix that contains attention scores. Attention scores in z inform the model of what elements are more important and what embedding information is more important. Multi-head attention can exploit GPU power by parallelizing heads, which is prominent in its appeal.

When dividing X_e between heads, it is commonly divided along the d dimension — resulting in a $(l, \frac{d}{h})$ dimensional space for each head, allocating a portion of the embedding space to each head. The input to each head is duplicated thrice, forming the values, keys, and queries. Each value, key, and query matrix first undergo a linear transformation:

$$\begin{aligned} V &= W_V X_e + b_V, \\ K &= W_K X_e + b_K, \\ Q &= W_Q X_e + b_Q, \end{aligned} \tag{2.8}$$

where W are learned weight matrices and b are learned bias vectors — values, keys, and queries have separate weights and biases, as denoted by the subscripts. Reference to heads is removed from the equations for simplicity. The values, keys, and queries are named as such because the structure of the multi-head attention is analogous to that of a database retrieval system. The naming is representative of how they interact with each other. It is hypothesized that they could hold information representative of their named functions since their interactions will affect their learned weights. Following the linear transformation, attention is computed using the values, keys, and queries. The attention function most commonly used is scaled dot-product attention, which will be the attention function used in this thesis. Scaled dot-product attention takes the form:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T}{\sqrt{d/h}})V, \tag{2.9}$$

$$\text{Softmax}(a) = \frac{\exp(a)}{\sum_{j=1}^{K} \exp(a)}. \tag{2.10}$$

An upper left triangle mask is applied to QK^T to prevent the queries from attending to keys that occur later in the sequence. The output of the attention function for each head is then concatenated together, followed by a linear transformation, producing the resultant matrix z:

$$z = W_A A + b_A, \tag{2.11}$$

where A is the output of the attention function in (2.9).

2.5 Normalization

Introduced to mitigate the presence of internal covariate shift in neural networks, batch normalization ([Ioffe and Szegedy, 2015]) is used between layers of the encoder. Batch normalization decreases training time by allowing the user to set higher learning rate values and mitigates the variance of random initialization of weights. Batch normalization uses mean-standard deviation normalization:

$$n_i' = \frac{n_i - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{2.12}$$

where n_i is the value at the i'th neuron, μ is the mean of the batch, σ^2 is the variance of the batch, and ϵ is a small constant value added to the variance for numerical stability.

Layer normalization ([Ba et al., 2016]) was introduced shortly following the discovery of batch normalization. Layer normalization is the transpose of batch normalization, computing mean and variance across all neurons in the layer within a single training case

rather than an entire batch. While the original Transformer uses layer normalization for their normalization layers, this study uses batch normalization in this study. The superiority of layer normalization in the Transformer can be mainly attributed to the varying sequence lengths found in sentences ([Shen et al., 2020]). However, this does not apply to the datasets examined in this study. Additionally, batch normalization can help dampen outliers found in time series ([Zerveas et al., 2021]), which is not an issue in natural language processing.

2.6 Reinforcement Learning

Reinforcement learning (RL) is a collection of optimization algorithms where an agent aims to learn the optimal strategy for interacting with its environment based on trial-and-error learning [Sutton and Barto, 2018]; the environment describes the world with which the agent interacts. Reinforcement learning algorithms can be categorized into model-based or model-free algorithms; in model-based approaches, the agent uses a predictive model of the environment to determine the possible outcomes of its actions. Model-free approaches, which don't use predictive models of the environment to guide the agent's decision-making process, rely more on the expected returns of their actions to understand their environment. Model-free reinforcement learning can be divided into two main categories: value-based and policy-based. Value-based approaches aim to learn or estimate the value or expected return of state-action pairs. In policy-based approaches, the agent devises a policy such that the action performed in every state helps the agent gain maximum future reward.

At each time step t, the agent receives observations from the environment and takes actions based on those observations. The environment then responds with a new state and a reward signal, which the agent uses to update its policy or value function. This loop continues until the agent reaches its goal or the environment ends. Using a neural network, the agent can progressively learn a model of the optimal policy or value function and eventually be capable of taking actions that would earn the largest long-term return.

A Markov decision process can be used to model reinforcement learning problems. This consists of four elements: the state space of the environment s, the action space of the environment a, the reward function r, and the transition probability p. The state space encompasses any information from the agent's environment that can and should be used to make decisions. The actions space includes the agent's decision variables — the set of all possible actions. The reward function defines the reward given to the agent after an action is executed and the results of the action are realized. Rewards can also have zero or negative values, often denoting punishment. Lastly, the transition probability delineates the stochasticity of the environment: the probability of a specific state s_{t+1} being observed when action a_t is taken at a specific state s_t. If the probability is 1, the environment would be deterministic.

2.6.1 Multi-agent RL

Multi-agent reinforcement learning (MARL) extends the principles of reinforcement learning to environments containing multiple agents that learn simultaneously. In such settings, each agent must not only optimize its own policy with respect to the environment, but also account for the behaviors of other agents, which may be cooperative, competitive, or a mix of both. At each timestep, all agents select actions $a_1, a_2, \ldots, a_n$, resulting in a transition to the next state and the assignment of rewards to each agent. Agents may share the same reward function in cooperative scenarios, have opposing reward functions in competitive scenarios, or occupy a general-sum environment with mixed incentives.

This introduces several unique challenges that distinguish MARL from the single-agent case. Among these is the issue of non-stationarity: from the perspective of any one agent, the environment appears to change over time as the policies of other agents evolve during training. This violates the stationary assumptions under which many reinforcement learning algorithms were designed, making convergence more difficult. Additionally, partial observability can significantly increase the complexity of many environments, as agents may have limited visibility into the actions or states of other agents, making coordination and decision-making more challenging.

Approaches to MARL can be broadly categorized into centralized and decentralized methods. In fully centralized training, a joint policy is learned over the combined action space of all agents, but this quickly becomes intractable as the number of agents grows. Decentralized methods, on the other hand, train each agent to act independently, often using only local observations; while scalable, such methods struggle to achieve coordination without additional mechanisms. A common compromise is centralized training with decentralized execution (CTDE), where agents share information during training but act independently at test time. Techniques such as value decomposition, actor-critic extensions, and attention-based communication have been developed to improve performance in this setting.

2.6.2 Value-Based RL

Value-based reinforcement learning seeks to estimate a value function that captures the expected return from a given state or state–action pair, with the optimal policy obtained by selecting actions that maximize these estimates [Sutton and Barto, 1998]. Early formulations such as Q-learning [Watkins and Dayan, 1992] provided an off-policy, model-free update rule grounded in the Bellman optimality equation, while SARSA [Rummery and Niranjan, 1994] offered an on-policy alternative. The introduction of deep neural networks enabled algorithms such as the Deep Q-Network (DQN) [Mnih et al., 2015] to achieve human-level performance in discrete-action domains, with stabilizing techniques including experience replay and target networks. Subsequent refinements addressed known limitations: Double DQN mitigated overestimation bias [Van Hasselt et al., 2016], the Dueling architecture separated state and advantage estimation for improved efficiency [Wang et al., 2016], and Rainbow integrated multiple enhancements into a single framework [Hessel et al., 2018]. While highly effective in large discrete spaces, these approaches

encounter scalability challenges in continuous control tasks, where action maximization becomes computationally demanding.

2.6.3 Policy-Based RL

Policy-based methods directly parameterize the policy $\pi(a|s;\theta)$ and optimize it with respect to expected return, avoiding the need for explicit action-value estimation [Sutton and Barto, 1998]. The REINFORCE algorithm [Williams, 1992] formalized this approach using Monte Carlo return estimates, though high variance in gradient estimates can slow convergence. Actor–Critic architectures [Konda and Tsitsiklis, 1999] addressed this by incorporating a learned value function as a baseline, reducing variance while maintaining unbiasedness. To improve stability in high-dimensional policy spaces, Trust Region Policy Optimization (TRPO) [Schulman et al., 2015] introduced a constrained optimization framework, later simplified in Proximal Policy Optimization (PPO) [Schulman et al., 2017] through a clipped surrogate objective. Deterministic Policy Gradient (DPG) [Silver et al., 2014] and its deep variant Deep Deterministic Policy Gradient (DDPG) [Lillicrap et al., 2015] extended policy-gradient methods to continuous action spaces, offering low-variance updates at the cost of potential exploration inefficiency. These methods are particularly well-suited to environments with continuous or stochastic control, though their performance can be sensitive to hyperparameter tuning and exploration strategies.

2.6.4 Model-Based RL

Model-based approaches learn an explicit model of the environment's transition dynamics, and in some cases the reward function, enabling the agent to perform planning through simulated rollouts [Sutton, 1991, Moerland et al., 2023]. The Dyna architecture [Sutton, 1991] exemplified early integration of model learning with value updates, while more recent methods have emphasized sample efficiency in continuous domains. PILCO [Deisenroth and Rasmussen, 2011] leveraged Gaussian process dynamics models for data-efficient policy search, though scalability to high-dimensional observations remains limited. Neural network dynamics models combined with Model Predictive Control have achieved competitive robotic performance [Nagabandi et al., 2018], and latent-space models such as World Models [Ha and Schmidhuber, 2018] demonstrated the potential of compact state representations for imagination-based planning. Building on this, the Dreamer family of algorithms [Hafner et al., 2019, 2020] employed recurrent state-space models to generate latent trajectories, achieving strong results in both discrete and continuous control benchmarks. While model-based RL often outperforms model-free methods in terms of sample efficiency, performance is highly dependent on model fidelity, with compounding errors over long horizons posing a persistent challenge.

Chapter 3

ACOPF with GNN for Warm-Start Optimization

3.1 Introduction

This chapter represents the first stage of the research journey: the use of deep learning to solve complex real-world problems in power systems. This is a theme that inspired the work presented in later chapters. Building on the background presented in Chapter 2, it applies deep reinforcement learning (specifically PPO) in combination with Graph Neural Networks (GNNs) to address the long-standing challenge of efficiently solving Alternating Current Optimal Power Flow (ACOPF).

Optimal power flow (OPF) is a fundamental task in power system management that seeks to identify an efficient, secure, and cost-effective operating point for an electrical grid. This is achieved by adjusting control variables, such as generator outputs and voltage settings, to minimize an objective function (e.g., operating cost or losses) while satisfying physical and operational constraints, including power balance, voltage limits, and line flow limits. Given a power system where $\mathscr{N}$ denotes the set of all buses, $\mathscr{G}$ the set of all generators, and $\mathscr{L}$ set of all transmission lines, the ACOPF is formulated as a set of equality and inequality constraints and an objective function to be minimized [Wood et al., 2013, Frank et al., 2012, Low, 2014]. Formally,

minimize:

$$\sum_{i \in \mathscr{G}} (C_{2i} P_{gi}^2 + C_{1i} P_{gi}) \tag{3.1}$$

Subject to:

$$P_{gi}^{min} \leq P_{gi} \leq P_{gi}^{max}, \qquad \forall i \in \mathscr{G}, \tag{3.2}$$

$$Q_{gi}^{min} \leq Q_{gi} \leq Q_{gi}^{max}, \qquad \forall i \in \mathscr{G}, \tag{3.3}$$

$$V_i^{min} \leq V_i \leq V_i^{max}, \qquad \forall i \in \mathscr{N}, \tag{3.4}$$

$$|S_{flow,ij}| \leq |S_{flow,ij}^{max}|, \qquad \forall (i,j) \in \mathscr{L}, \tag{3.5}$$

$$P_i = \sum_{\forall j \in \mathscr{N}} V_i V_j (g_{ij} cos(\theta_j - \theta_i) + b_{ij} sin(\theta_j - \theta_i)), \quad \forall i \in \mathscr{N}, \tag{3.6}$$

$$Q_i = \sum_{\forall j \in \mathscr{N}} V_i V_j (g_{ij} sin(\theta_j - \theta_i) - b_{ij} cos(\theta_j - \theta_i)), \quad \forall i \in \mathscr{N}, \tag{3.7}$$

where $C_{1.}$ and $C_{2.}$ are coefficients related to the costs of power generation, P_i and Q_i are the total real and reactive power at bus $i \in \mathscr{N}$, respectively, P_{gi} and Q_{gi} are the total real and reactive generation at generator $i \in \mathscr{G}$, $S_{flow,ij}$ is the apparent power of line $(i,j) \in \mathscr{L}$, V_i and θ_i are the voltage magnitude and the phase angle at bus $i \in \mathscr{N}$, respectively, g_{ij} and b_{ij} are the conductance and susceptance, respectively, of the line $(i,j) \in \mathscr{L}$ that connects buses i and j. 3.1 is the objective function to be minimized, relating to power generation costs. 3.2-3.5 are inequality constraints of the ACOPF related to limits on power generation, voltage limits, and line flow limits. 3.6 and 3.7 are equality constraints to enforce real and reactive power balance.

OPF is an NP-hard problem in its original form due to the nonlinear and nonconvex nature of the AC power flow equations [Lavaei and Low, 2012]. These properties give rise to multiple local optima and make it computationally challenging to guarantee convergence to a globally optimal solution, particularly for large-scale power systems. The alternative to solving ACOPF is to apply a direct current (DC) approximation using a relaxation that negates the reactive component(s) of the power system — thus making the optimization problem linear and convex. Direct Current Optimal Power Flow (DCOPF) is a common approach for solving OPF due to the lack of computational complexity associated with solving it. Nonetheless, the approximation of OPF's true form will incur significant monetary losses due to suboptimal solutions obtained using DCOPF [Baker, 2021].

AC optimal power flow (ACOPF) algorithms are commonly evaluated using standardized benchmark power system models implemented in software toolboxes such as MATPOWER and PYPOWER, together with the Institute of Electrical and Electronics Engineers (IEEE) power system test cases. The IEEE test cases provide synthetic but realistic transmission network models of varying size and complexity, specifying bus, generator, load, and transmission line parameters. These cases are not intended to represent any specific real-world grid, but rather to serve as controlled benchmarks for algorithmic evaluation.

MATPOWER [Zimmerman et al., 2011] a widely used MATLAB-based package for power flow and optimal power flow analysis that implements established numerical solvers, including interior-point methods for ACOPF. PYPOWER [Lincoln and Zimmerman, 2011] is a Python reimplementation of MATPOWER that follows the same modeling conventions, data formats, and optimization formulations. Together, these tools are commonly used in the literature as simulation environments for testing and comparing ACOPF solution methods under consistent conditions, allowing researchers to assess convergence behavior, solution quality, and computational performance across different network sizes.

In this work, the objective is to solve the ACOPF problem efficiently across a range of IEEE benchmark systems, using synthetic training data generated from these standardized test cases [Christie, 1999]. Rather than replacing existing optimisation methods, the goal is to learn a model that produces high-quality initial starting points for a conventional ACOPF solver. This approach is well suited to the considered setting because modern interior-point solvers are known to reliably recover optimal solutions when they converge within the allotted time [Nocedal and Wright, 2006]. Consequently, the primary challenge is not solution accuracy, but computational efficiency. This motivates learning initialisations that improve convergence speed, thereby reducing overall solve time while preserving the accuracy guarantees of the underlying optimisation algorithm.

In the last two decades, finding direct solutions for ACOPF has become a popular area of research [Mohagheghi et al., 2018]. Proposed solutions often fall short because they cannot consistently produce feasible solutions, achieve global optimum, or address space-time complexity[1] issues related to solving a nonconvex optimization problem.

Deep learning approaches to solving ACOPF have been well explored (see, e.g., [Zhou

[1]Space-time complexity refers to an algorithm's efficiency in computation time and required memory.

et al., 2020b, Zhang and Zhang, 2022, Wang et al., 2022, Nellikkath and Chatzivasileiadis, 2022]). While deep learning has been shown to compute feasible solutions quickly, they are consistently far from global optimum relative to other methods, such as iterative solving methods. Iterative solvers, such as gradient descent, Newton-Raphson's method, or quasi-Newton methods, are numerical methods that use an initial value to generate a sequence of increasingly accurate approximate solutions for a specific class of problems. In these methods, the n-th approximation is derived from the previous approximations, gradually refining the solution until it reaches a global optimum. Due to neural networks' inherent functionality for generalization, it would be nearly impossible to generate a truly optimal output. The only means of consistently producing optimal solutions would be to create an architecture designed to overfit and then train it on every possible combination of inputs, which is infeasible for a continuous, thus infinitely large, input space. Alternatively, while global optimality is not necessarily guaranteed every time, iterative solvers alone are generally capable of consistently producing optimal ACOPF solutions that are better solutions than other researched methods, such as deep learning, but this comes at the cost of computation time; convergence to the optimum can often not be done in real-time, especially in larger power systems. This problem can be mitigated if the iterative solvers are informed with an accurate initial guess or warm-start. Providing a warm-start can significantly improve the convergence rate and reduce the number of iterations required to converge [Cao et al., 2023].

This chapter proposes a novel time-efficient solution to ACOPF using deep reinforcement learning in conjunction with iterative solving methods, specifically, the interior point solver (IPS). The deep reinforcement learning agent is not trained to produce a solution to ACOPF that is feasible or cost-minimizing; instead, the agent is trained to compute an initial guess that minimizes the time it would take for the IPS to find a solution. Due to the nonconvex properties of ACOPF, an initial guess that is nearer to the global optimum in terms of costs and feasibility may be very far in proximity to the global optimum. As a result, training an agent to generate initial guesses that prioritize feasibility and cost minimization may actually lead to longer IPS convergence times.

Training the agent to make initial guesses that minimize IPS computation time requires a feedback loop to inform the loss function of the quality of the initial guess based on the amount of time it takes for the IPS to solve the ACOPF. This approach should employ reinforcement learning, as it would not be a practical way to generate a predefined dataset for traditional machine learning approaches; to do so would require generating thousands or even millions of datapoints with random inputs and their resulting IPS convergence times as labels. This method would result in a heavily imbalanced dataset of bad initial guesses, whereas with reinforcement learning, the initial guesses will start out random but as they improve the agent will be learning from better data, from which it can improve even further.

Our proposed framework uses proximal policy optimization (PPO), a type of deep reinforcement learning algorithm, to train a neural network to compute an initial guess of the ACOPF solution, allowing IPS to solve it in significantly fewer iterations, thus decreasing the computation time. Reinforcement learning agents learn via trial and error as they interact with their environment, take actions, and observe the rewards earned from making particular actions. PPO is a simple actor-critic reinforcement learning algorithm that seeks to find an optimal policy (a function that maps observations to actions)

rather than assigning values to state-action pairs [Schulman et al., 2017]. This is coupled with a GNN to act as the agent's learning mechanism — exploiting the graph structure of the power grid. GNNs are a specialized type of neural networks that are well-suited to process graph-structured data. By analyzing and utilizing the patterns and relationships within the graph, GNNs can provide more accurate predictions about the entities involved compared to models that only consider individual entities in isolation, such as a standard multi-layer perceptron (MLP). Using a GNN may allow for adaptation to changes in topology with only minor adjustments. In contrast, an MLP, the standard in deep learning-based ACOPF literature, would require a complete retrain. Topology adaptability is essential in real-world scenarios where line and generator outages or other unexpected changes to topology can occur. GNNs are capable of handling changes in topology because their weights are agnostic to changing input dimensions, unlike MLPs which would fail due to matrix shape mismatch. While this is a large draw of GNNs in this domain, it is important to note that major changes in topology would very likely still require full retrain of the GNN to perform well since a change in topology can significantly alter nodes' relationships with each other, and thus learned weights for one topology may not be transferable to other topologies without some amount of retraining.

One study has used GNN for ACOPF, [Liu et al., 2022c], which focused on computing ACOPF solutions directly with the GNN. Although warm-start solutions for iterative solvers using machine learning and deep learning have been explored in [Cao et al., 2023], [Baker, 2019b], [Pan et al., 2023], [Canyasse et al., 2017], [Donti et al., 2021] these methods do not use reinforcement learning to factor in the convergence time of the employed iterative solver into the loss function of the learning algorithm and all but one [Diehl, 2019] do not use GNN to take advantage of the graph structure of the power system.

With the proposed framework, I aim to demonstrate a middle-ground solution to ACOPF that combines the speed of a trained neural network with the accuracy of iterative solvers. This framework is evaluated on the feasibility of solutions and the computation time required to compute solutions. Experiments are conducted using standard IEEE test environments.

The contributions of this work are as follows:

1. I create a novel framework for solving ACOPF utilizing GNN and PPO as learning mechanisms.
2. I introduce a novel deep reinforcement learning reward function for ACOPF that considers the iterative solver's convergence time, which has not yet been explored in ACOPF literature.
3. I demonstrate state-of-the-art results that do not compromise accuracy or computation time.

By showcasing this novel solution to ACOPF, I aim to provide a new direction for future ACOPF research to compute accurate solutions in real-time.

The structure of this chapter is as follows: Section 2 presents the related work; Section 3

outlines the methodology, including the characteristics of the agent, the GNN architectures, and the structure of the training algorithm; Section 4 provides an overview and discussion of the results obtained from each experiment; and Section 5 provides a brief conclusion of the chapter and description of future work.

3.2 Related Work

The Optimal Power Flow problem has been extensively studied since its original formulation by Carpentier in 1962 [Carpentier, 1962]. It aims to determine a steady-state operating point that minimizes generation costs while satisfying operational and security constraints. Over the years, the literature has proposed a wide range of solution methods, from classical deterministic approaches to more recent heuristic and learning-based frameworks. Recent comprehensive reviews [Risi et al., 2022, Khaloie et al., 2025] highlights this evolution and categorizes OPF solution methods into classical, probabilistic, metaheuristic techniques, and machine learning-based techniques.

3.2.1 Classical Optimization Methods

Early OPF methods were deterministic and gradient-based, employing techniques such as Newton–Raphson and Gauss–Seidel load flow solvers [Dommel and Tinney, 1968, Momoh et al., 1999]. The DCOPF formulation, a linear approximation of AC power flow, is still widely used for its computational tractability in electricity markets and contingency analysis. However, DCOPF neglects voltage magnitudes, reactive power, and line losses, which can result in infeasible solutions under AC constraints [Bukhsh et al., 2013]. ACOPF, on the other hand, captures full non-linear system physics but introduces non-convexities that increase the risk of convergence to local minima [Hiskens, 2001]. Interior-point methods and sequential quadratic programming have been employed to improve scalability, yet these methods remain computationally intensive for large-scale networks.

3.2.2 Probabilistic and Stochastic Approaches

With the rapid integration of renewable energy sources (RES) and distributed generation, accounting for uncertainty has become essential. Probabilistic load flow methods, first proposed in the 1970s [Borkowska, 1974], model uncertain inputs via probability density functions. Monte Carlo simulations [Su, 2005, Papaefthymiou et al., 2006] and analytical approaches [Allan et al., 1991] have been widely applied, although Monte Carlo methods are computationally expensive. Fuzzy set theory [Das et al., 1999, Bijwe and Raju, 2006, Kalesar and Sei, 2010] and cumulant-based techniques [Hossain et al., 2019] provide alternatives with reduced complexity.

3.2.3 Metaheuristic Optimization Techniques

Metaheuristic methods have become increasingly popular due to their ability to handle non-convex OPF formulations. Examples include Genetic Algorithms [Attia et al., 2012, Kahourzade et al., 2015], Particle Swarm Optimization [Kalfallah et al., 2015, Somasundaram et al., 2004], Differential Evolution [Varadarajan and Swarup, 2008], Artificial Bee Colony [Ayan et al., 2014, Adaryani and Karami, 2013], Gravitational Search Algorithm [Duman et al., 2012], and Grey Wolf Optimizer [El-Fergany and Hasanien, 2015]. These algorithms are population-based and inspired by biological or physical processes, enabling effective exploration of large solution spaces and avoidance of local optima. However, they often suffer from high computational cost and lack guaranteed optimality, limiting their real-time application.

3.2.4 Machine Learning Approaches for Optimal Power Flow

Recent work has increasingly applied machine learning to the OPF problem, with Khaloie et al. [Khaloie et al., 2025] providing a comprehensive taxonomy. These approaches can be grouped into four main categories: supervised learning, unsupervised learning, reinforcement learning, and warm-start methods. Each provides unique strengths in terms of computational efficiency, adaptability, and feasibility.

Supervised Learning

In supervised learning, models are trained on labeled datasets that pair system inputs (e.g., load profiles, RES injections) with optimal OPF solutions (e.g., generator setpoints). This direct mapping is typically formulated as a regression problem, and can be categorized as unconstrained, constrained, or graph-based learning [Khaloie et al., 2025]. Unconstrained models such as feed-forward Neural Networks [Zhou et al., 2023], convolutional neural networks [Yang et al., 2021, Jia et al., 2023], and random forest [Rahman et al., 2021] predict OPF solutions quickly but risk feasibility violations. Constrained models embed operational limits (e.g. line flow and voltage constraints) via penalty methods or implicit layers [Khazaei and Zhao, 2022, Wu et al., 2022, Liu et al., 2022a, 2021b], ensuring greater reliability. More recently, GNNs have been used to leverage system topology, improving scalability and adaptability to network reconfigurations [Hansen et al., 2022, Liu et al., 2022b, 2021c, Gao et al., 2023].

Unsupervised Learning

Unsupervised learning for OPF remains limited but has been explored through Generative Adversarial Networks (GANs). GANs learn the distribution of OPF solutions without explicit labels, generating synthetic solutions that mimic real optimal dispatches [Goodfellow et al., 2014]. Conditional GANs further incorporate feasibility constraints, ensuring generated solutions respect AC or DCOPF limits [Li et al., 2022, Wang and Srikantha,

2022]. While promising, the lack of explicit control over feasibility remains a key challenge for GAN-based OPF mapping.

Reinforcement Learning

Reinforcement learning (RL) formulates OPF as a sequential decision-making problem within a Markov Decision Process, where an agent interacts with a simulated power system environment and learns a policy to balance operating costs, system security, and constraint satisfaction [Yang et al., 2020, Yu et al., 2021]. In these approaches, the system state typically includes network conditions such as bus voltages, loads, and generation levels, while the agent's actions correspond to control variables such as generator set-points or voltage magnitudes.

Deep RL methods have been applied to both AC and DCOPF using various algorithms One line of work applies actor–critic methods such as deep deterministic policy gradient (DDPG) [Nie et al., 2019, Yan and Xu, 2020, Xiao et al., 2023] to learn direct mappings from system states to OPF solutions. For example, [Nie et al., 2019] propose a DDPG-based approach that outputs generator control actions in continuous action spaces, demonstrating near-optimal performance on small benchmark systems such as the IEEE 9-bus case. Twin delayed deep deterministic policy gradient (TD3) is also well explored for solving ACOPF [Zhen et al., 2022, Woo et al., 2020, Wu et al., 2023], Specifically, [Zhen et al., 2022] applies a Twin Delayed Deep Deterministic Policy Gradient (TD3) approach to ACOPF, where a neural network policy directly outputs control actions while incorporating uncertainty through stochastic load variations. Their method is evaluated on larger benchmark systems, such as the IEEE 118-bus case, and opt for constraint satisfaction through reward design. Similarly, soft actor-critic (SAC) methods have been applied to ACOPF [Han et al., 2023c, Sayed et al., 2022, Wang and Tang, 2022, Sayed et al., 2023], where stochastic policies are learned to improve exploration and stability during training. These approaches also learn direct mappings from system states to control actions, but, as with other RL-based methods, constraint satisfaction is typically handled implicitly through reward design rather than being explicitly guaranteed. Proximal policy optimization (PPO) has also been explored for OPF problems [Zhou et al., 2020a, 2022b, López-Cardona et al., 2022], where policies are trained using clipped policy updates to improve stability. These methods similarly aim to directly approximate OPF solutions or control actions, achieving fast inference but facing the same challenges in ensuring feasibility and scalability to larger systems. These methods generally aim to learn policies that directly output control actions or approximate OPF solutions without relying on traditional optimisation solvers at inference time.

In contrast to these approaches, my proposed method does not aim to replace the optimisation solver or directly approximate the OPF solution. Instead, it uses reinforcement learning to generate initial starting points for a conventional solver, allowing the approach to retain feasibility and optimality guarantees while improving computational efficiency.

Warm-Start Methods

Warm-starting employs ML predictions to provide high-quality initializations for classical solvers, significantly accelerating convergence [Cao et al., 2023, Baker, 2019b, Pan et al., 2023, Canyasse et al., 2017]. Techniques include NN-based regressors [Sadat and Sahraei-Ardakani, 2021], Random Forests [Baker, 2019a], and GNN-based warm-starts [Diehl, 2019]. Studies show that ML warm-starts can outperform conventional DCOPF-based or flat-start initializations, reducing runtime and improving feasibility. For example, a GNN warm-start for AC-OPF reduced runtime compared to both traditional solvers and DCOPF initializations [Diehl, 2019]. Despite their promise, warm-starts must be carefully designed to ensure robustness across varying system states and contingencies.

3.2.5 Summary

The OPF literature spans a broad spectrum of methodologies, evolving from classical deterministic solvers to modern learning-based frameworks. Classical optimization methods, while theoretically well-grounded, face scalability and non-convexity challenges in realistic AC formulations. Probabilistic and stochastic approaches incorporate uncertainty from renewable generation and demand, but often incur high computational costs. Metaheuristic algorithms offer flexibility for non-convex search spaces and complex constraints, yet their iterative, population-based nature limits real-time applicability.

Machine learning techniques have introduced new possibilities for accelerating OPF and improving adaptability to changing grid conditions. Supervised learning enables fast, direct mappings from system states to OPF solutions, though ensuring feasibility remains a key concern. Unsupervised methods, particularly GAN-based approaches, offer data-driven generative modeling but still lack robust feasibility guarantees. Reinforcement learning provides a sequential decision-making framework capable of handling multi-objective trade-offs, scalability to large grids remain open challenges, and consistent with other non-hybrid machine learning-based methods, struggles with ensuring feasible solutions. Hybrid warm-start strategies bridge learning-based prediction and traditional solvers, achieving significant runtime improvements while leveraging the reliability of established optimization routines.

Overall, each class of methods offers distinct advantages and trade-offs. Hybrid warm-start approaches that combine the accuracy and feasibility guarantees of classical solvers with the speed and adaptability of learning-based models represent a promising direction for future OPF research; which is why I choose to explore this further. Compared to existing methods, I employ several methods that are novel or under-researched to improve performance with improvements, namely, incorporating IPS convergence time into the agent's loss function, and the use of GNNs to exploit the graph structure of the power system.

3.3 Background

3.4 Proximal Policy Optimization

As a derivative of policy gradient and trust region methods, PPO is a policy-based reinforcement learning algorithm that employs elements from each to reach a balance between ease of implementation, sample complexity, and ease of tuning to compute an update that minimizes the cost function while ensuring the deviation from the previous policy is relatively small [Schulman et al., 2017]. PPO is used because policy-based methods generally work better in environments with infinitely large state and action spaces, as value-based methods cannot reasonably obtain accurate mappings of optimal state-action pairs without significant training.

Algorithm 1 outlines PPO's structure. In each episode of training, for T time steps (where T is a much smaller value than the length of an episode [2]), the agent will use the same policy; once the number of time steps reaches T, the agent will undergo a policy update. PPO uses an actor-critic structure: the actor learns the optimal policy, whereas the critic estimates the value function, which evaluates the expected return of a given state. The actor and the critic are both modeled by a neural network.

Algorithm 1 PPO

for iteration $= 1, 2, \ldots$ do
  Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
  Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  Optimize surrogate L wrt θ, with K epochs and
  minibatch size $M \leq NT$
 $\theta_{\text{old}} \leftarrow \theta$
end for

The objective function to be minimized by the agent is defined as:

$$L_t(\theta) = L_t^{clip}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S\left[\pi_\theta\right](s_t)\,, \tag{3.8}$$

where c_1 and c_2 are user-defined coefficients to determine the weight of each component in the objective function, $L_t^{VF}(\theta)$ is the loss of value function (the critic network), L^{clip} is the clipped surrogate loss, and S is the entropy of the policy. The policy, π_θ, is defined by a multivariate Gaussian distribution $\pi_\theta \sim Norm\left(\boldsymbol{\mu}_\theta(s_t), \boldsymbol{\Sigma}_{\pi\theta}\right)$. The mean of the distribution, $\boldsymbol{\mu}_\theta(s_t)$, is the output produced by the actor network, and $\boldsymbol{\Sigma}_{\pi\theta}$ is the diagonal matrix of the covariance matrix with a user-defined standard deviation; the standard deviation is set relatively high for earlier episodes and decays as the episodes progress, adding small amounts of random noise to the action and aiding in exploration.

[2] An episode refers to a series of agent-environment interactions between the initial state and the terminal state.

The clipped surrogate loss is defined as

$$\begin{aligned} L^{clip}(\theta) &= \min(R_t(\theta)\hat{A}_t^{\pi_\theta,\,\text{old}}\,, \\ &\quad \text{clip}\left(R_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t^{\pi_\theta,\,\text{old}}\,), \end{aligned} \tag{3.9}$$

where θ_{old} refers to the policy parameter before the actor is updated, $\hat{A}_t$ is the advantage estimate, which is defined as:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \tag{3.10}$$

where

$$\delta_t = R_t + \gamma V\left(s_{t+1}\right) - V\left(s_t\right), \tag{3.11}$$

and $R_t(\theta)$ is the ratio between the new and old policies, defined as:

$$R_t(\theta) = \pi_\theta\left(a_t \mid s_t\right)/\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right). \tag{3.12}$$

The clip function ensures that $R_t(\theta)$ is between $1-\varepsilon$ and $1+\varepsilon$, where ε is a user-defined parameter to determine the tightness of the clip boundaries. In (11), variables γ and λ are user-defined variables that control the reward discount; larger values allow the agent to consider long-term rewards, while smaller values enable the agent to consider shorter-term rewards, and $V(\cdot)$ is the value function modeled by the critic network.

The agent's training generally concludes when the objective function (9) reaches a point of convergence, and the loss is no longer decreasing.

It is important to note that, in contrast to standard reinforcement learning settings, the formulation used in this work is not sequential. Each ACOPF problem instance is treated as an independent state, and the policy is applied once to produce a full solution (or warm-start) in a single time step. As such, there is no temporal evolution, no multi-step trajectory, and no dependence between successive states. As a consequence, elements of PPO designed for multi-step temporal credit assignment, such as the advantage estimation in 3.10 and λ-returns in 3.11, reduce to simplified forms in this setting. Since each decision consists of a single time step ($\lambda = 0$), there is no temporal accumulation of rewards, and the advantage estimate depends only on the immediate reward and value prediction.

It is also important to note that, due to the single-step nature of the problem formulation, the role of the critic differs from that in standard reinforcement learning settings. In multi-step problems, the critic is used to estimate long-term returns and enable temporal credit assignment. However, in this work, each ACOPF instance is solved in a single time step, and there is no trajectory over which rewards are accumulated. As a result, the critic does not provide the same benefits as in sequential settings, since there is no notion of long-term return or temporal credit assignment. The learning problem instead reduces to a single-step decision, where the policy maps each independent system state directly to an action based solely on the immediate reward.

In practice, we initially adopted a standard actor–critic formulation for consistency with PPO, but found that the critic plays a limited role in this setting, as the advantage reduces to a function of the immediate reward. The primary learning signal is therefore driven directly by the observed reward, and the policy effectively learns a one-step mapping from system states to high-quality initializations.

3.5 Methodology

Firstly, I will define the reinforcement learning problem by outlining the state space, reward function, and action space. The state of the environment s, the agent's input, includes all active and reactive load variables for all buses in the network that belong in $\mathscr{N} = \{1, \ldots, N\}$:

$$s = [P_{l1}, \cdots, P_{lN}, Q_{l1}, \cdots, Q_{lN}], \tag{3.13}$$

where P_{li} is the real power demand at bus i and Q_{li} is the reactive power demand at bus i. The agent will then execute an action using this state information to inform its decision. The action space is comprised of variables related to power generation (for all generators that belong in $\mathscr{G} = \{1, \ldots, G\}$), voltage magnitude, and voltage angles:

$$a = [P_{g1}, \cdots, P_{gG}, Q_{g1}, \cdots, Q_{gG}, V_i, \cdots, V_N, \theta_i, \cdots, \theta_N]. \tag{3.14}$$

Note that the agent's goal is not necessarily to produce a set of actions that result in a feasible and cost-minimizing solution to the ACOPF, so the reward function must reflect the agent's true goal to compute a set of actions that minimizes the amount of time required for the IPS to converge to a solution. Thus, I designed the following reward function:

$$r = \begin{cases} -r_{nc}, & \text{IPS could not converge given } a_t \\ -r_c, & \text{IPS converges given } a_t \end{cases}, \tag{3.15}$$

where r_{nc} is a fixed value denoting the penalty assigned to non-convergence, and r_c is the amount of time, in seconds, it takes for the IPS solver to converge given a. While non-convergence can result from truly unsolvable states, it is more commonly caused by poor initial guesses, as IPS can be highly sensitive to initialization. Thus, a negative reward is assigned when the IPS does not converge.

The value of r_{nc} must be chosen carefully to balance these objectives. If the penalty is too large, the agent may prioritize avoiding non-convergence at the expense of minimizing convergence time. Conversely, if it is too small, the agent may tolerate non-convergent solutions, particularly when convergence times are high.

In practice, the non-convergence penalty r_{nc} is chosen relative to the range of convergence times observed across the dataset. Specifically, r_{nc} is set to be more negative than the reward associated with the slowest successful convergence, ensuring that non-convergent solutions are always penalized more heavily. At the same time, the penalty is not set excessively large, allowing the agent to remain sensitive to differences in convergence

time among feasible solutions. This results in a balanced objective that encourages both convergence and reduced solver runtime.

3.5.1 Environmental Set-Up

The environment is constructed using PyPower, the official Python port of MATPOWER. Experiments are conducted using Python 3.7. computations are conducted on an NVIDIA A100 40GB GPU.

The proposed framework is trained and evaluated on the IEEE 14-bus, IEEE 118-bus, and IEEE 300-bus systems. As a baseline, results are first compared to PyPower's ACOPF solver, which initializes the optimization using a default heuristic (the midpoint between the minimum and maximum bounds of each decision variable). PyPower's IPS is used for all experiments, providing a consistent reference for convergence behavior.

In addition to this classical baseline, two learning-based approaches are considered. The first is a graph neural network (GNN) trained in a supervised manner to directly approximate feasible and cost-minimizing ACOPF solutions. This model learns a mapping from system states to solutions produced by the IPS, using a mean-squared error loss over a dataset of 20,000 samples. This baseline represents the common approach in the literature where neural networks are trained to replace or approximate OPF solvers.

The second baseline is a reinforcement learning model based on PPO using a multi-layer perceptron (MLP) instead of a GNN. In this case, the policy is trained with the same reward function as the proposed method, directly optimizing for reduced IPS convergence time. This baseline isolates the effect of the neural network architecture, as it does not exploit the graph structure of the power system.

These baselines also serve as an ablation study to validate the following claims for warm-start generation:

1. A loss function designed to minimize IPS convergence time is more effective than one designed to produce feasible and cost-minimizing solutions.
2. A neural network architecture that exploits the graph structure of the power system is more effective than one that treats nodes independently.

The structure of the training algorithm is as follows[3]: first, the load at each load bus is adjusted. This adjustment is done by randomly generating a value between 10% and 80% of the system generation limits for the total load across all load buses and then assigning a portion of that total load to each load bus using a Dirichlet distribution, which is defined as:

$$f(z, \boldsymbol{\alpha}) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{i=1}^{D} z_i^{\alpha_i - 1}, \tag{3.16}$$

[3]github.com/AzadDeihim/ACOPF-PPO-GNN

$$\mathrm{B}(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{D} \Gamma(\alpha_i)}{\Gamma\left(\sum_{i=1}^{D} \alpha_i\right)}, \tag{3.17}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^D$ is a vector of concentration parameters for each load bus, and D is the number of loads in the power network. The Dirichlet distribution is a distribution over a vector $z \in \mathbb{R}^D$ such that each $z_i > 0$ and:

$$\sum_{i=1}^{D} z_i = 1. \tag{3.18}$$

The vector returned by Dirichlet distribution denotes the share of the total real power demand each load bus will be delegated. The reactive power demand at each load bus is calculated via a randomly generated power factor between 0.8 and 1.

Given the new state of the environment, the agent will use these values to compute an initial guess. Then, a timer is started, and the initial guess is given to the PyPower IPS as a starting point. When the IPS converges, the timer is stopped. If it converges, the negative of the elapsed time is given as a reward. If it does not converge, a large negative reward is given. After 1,000 iterations of this process, the agent will undergo a policy update and then continue from the first step. Statistics regarding average reward, IPS convergence time, and the number of IPS non-convergences are recorded during each 1,000 time step period. The total number of iterations is not predefined; training will end once the agent's reward converges and is no longer improving. A flowchart diagram of the training algorithm is shown in Figure 3.1.

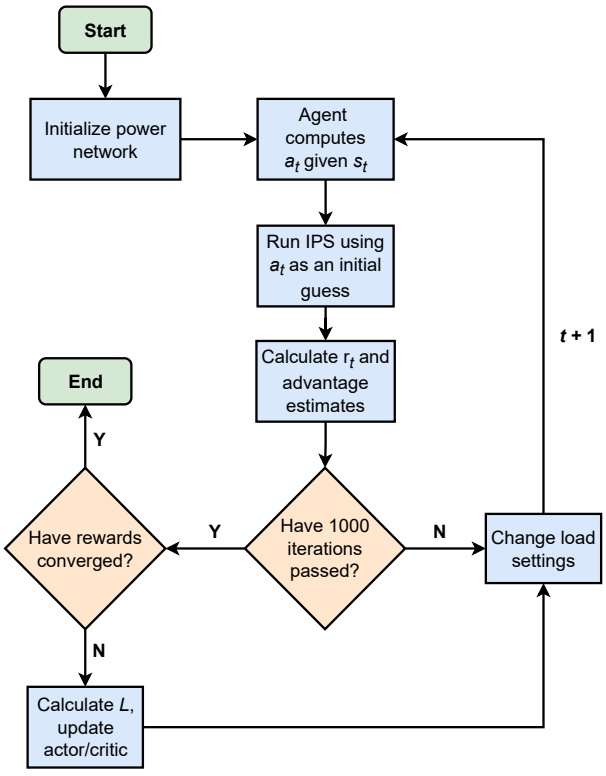

Figure 3.1: A flowchart of the training algorithm.

Table 3.1: Layer sizes for each neural network architecture. These values denote the dimensionality of the output of each layer, determined by the shape of the weight matrices between each layer.

	14-Bus		118-Bus		300-Bus	
GNN Layer	Actor	Critic	Actor	Critic	Actor	Critic
Layer 1 Size	128	128	256	256	512	512
Layer 2 Size	256	256	512	512	1024	1024
Layer 3 Size	128	128	256	256	512	512

	14-Bus		118-Bus		300-Bus	
MLP Layer	Actor	Critic	Actor	Critic	Actor	Critic
Layer 1 Size	128	128	256	256	512	512
Layer 2 Size	256	256	512	512	1024	1024
Layer 3 Size	128	128	256	256	512	512

Separate GNNs model the actor and critic, and their architecture only varies slightly. While they are both given the same input, they will output different values: the actor will output $a \in \mathbb{R}^{2G+2N}$, the action, whereas the critic will output the value or expected reward of state transition. In both the 14-bus experiment and the 118-bus experiment, the actor and critic will have three graph convolution layers, where each layer performs the node-wise transformation described in (8). The output for all nodes of the final graph convolution is flattened into a vector and then linearly transformed via:

$$a = WX + b, \tag{3.19}$$

where W is the weight matrix and b is the bias vector. A hyperbolic tangent activation function is used between all layers of the GNNs. The optimization algorithm used in backpropagation is the Adam optimizer.

Details regarding hyperparameter settings for each experiment, including the architecture of the actors and critics, can be found in Tables 3.1 and 3.2. The action standard deviation is initially set to 0.5 and decays by 0.01 every 250 time steps until it reaches 0. I chose discount factors of 0.0 because actions in different time steps are completely independent, and state transitions are entirely stochastic; thus, long-term rewards should not matter to the agent.

Table 3.2: Coefficient settings for the objective function of PPO, learning rate settings for actor and critic networks, and discount factor settings.

	14-bus	118-bus	300-bus
c_1	0.5	0.5	0.5
c_2	0.01	0.01	0.01
γ	0.0	0.0	0.0
λ	0.0	0.0	0.0
r_{nc}	12	12	15
Learning rate	0.0005	0.0005	0.0005

3.6 Results

This section presents results from the experiments outlined in Section 3.5.1. PPO-GNN and the baseline are evaluated on the same 5000 randomly generated states for all experiments. During testing, the critic network is discarded as it is only required for training; all initial guess computation is conducted by the actor. All models will be evaluated on their ability to produce initial guesses that can minimize IPS convergence time and minimize the number of non-convergences. Before presenting the results, it is important to clarify how the training data is generated. During training, each episode is sampled from an independently generated ACOPF setup, rather than from a fixed finite training set. As a result, the learning curves report performance over continuously generated problem instances. This means that the agent is not repeatedly optimizing over the same set of examples, and a separate validation set is not required in the same way it would be for a supervised learning model trained on a fixed dataset.

3.6.1 14-bus

The IEEE 14-bus test case represents a small power system with 14 buses, five generators, 11 loads, and 20 lines. For the IEEE 14-bus system, I demonstrate that our proposed framework can significantly decrease IPS computation time while minimizing the number of non-convergences. Training details are provided in Figure 3.2. Figure 3.2 shows that after about 9,000 time steps, the agent converges to an average reward of -0.54 and is able to compute initial guesses that allow the IPS to solve the ACOPF in under 0.55 seconds while only causing the IPS to not converge in less than 0.1% of states. Results are compared to the baselines in Table 3.3. This shows that the proposed framework outperforms the baselines significantly in terms of IPS convergence time and the number of non-convergences. By using the proposed PPO-GNN-based framework to compute initial guesses for IPS, a 33% speed-up over the PyPower ACOPF solver, a 14% speed-up over the GNN, and a 13% speed-up over PPO-MLP are observed. The initial guesses supplied by PPO-GNN resulted in a 100.0% convergence rate on the test set. On the other hand, initial guesses provided by the baselines incurred significantly lower convergence rates. Results in Figure 3.2 and Table 3.3 indicate that the quality of the initial guess plays an important role in the IPS's ability to converge. In the early training time steps, convergence rates below 75% are seen. Also, over the course of training, it is observed that the average IPS convergence time decreases from 0.85 to 0.53 seconds, demonstrating that even in some of the earlier time steps where PPO-GNN was undertrained, the initial guesses still resulted in lower IPS convergence times than the baselines. Additionally, it only takes the GNN roughly eight milliseconds to compute an initial guess. The total training time for PPO-GNN in this experiment was roughly five hours, the majority of which was IPS-solving time.

Table 3.3: Comparison of the mean and standard deviation of IPS convergence times and convergence rate of IPS between PPO-GNN and the baseline on the 14-bus test case.

	Mean r_c	Standard Deviation r_c	Convergence %
PPO-GNN	**0.53 seconds**	**0.09 seconds**	**100.0%**
PyPower	0.79 seconds	0.19 seconds	96.5%
GNN	0.62 seconds	0.13 seconds	97.9%
PPO-MLP	0.61 seconds	0.09 seconds	96.2%

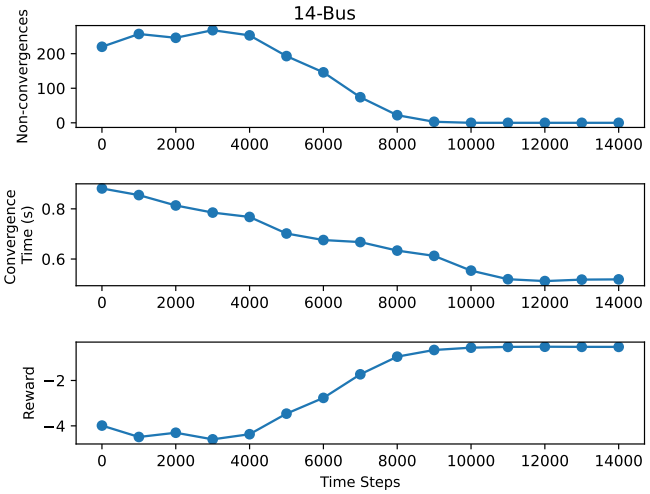

Figure 3.2: PPO-GNN training statistics for 14-bus test case. The three scatter plots represent the number of non-convergences, average IPS convergence time, and average reward on the y-axis and the number of time steps on the x-axis.

3.6.2 118-bus

The IEEE 118-bus test case represents a power system with 118 buses, 19 generators, 35 synchronous condensers, 177 lines, nine transformers, and 91 loads. Similar to the 14-bus system, our framework delivers improved performance over the baselines. Results of PPO-GNN are compared with the baselines in Table 3.4, demonstrating a 30% improvement over the PyPower solver, a 12% improvement over the GNN, and a 14% improvement over the PPO-MLP in terms of IPS computation time. Figure 3.3 shows a detailed visualization of training statistics. After about 12,000 time steps, the agent converged to an average reward of -1.19, an average IPS convergence time of 1.19 seconds, and a 100.0% convergence rate. Similar to the 14-bus experiment, PPO-GNN is the only model capable of achieving a 100.0% convergence rate on the test set. Also, during training, there are much lower convergence rates in the early stages of training, often below 60%, as well as IPS convergence times of over 2 seconds, showing that in this experiment, the outcome of the IPS is heavily reliant on the quality of an initial guess. Since the GNN was larger in this test case, it took the GNN about 18 milliseconds on average to compute an initial guess. The total training time for PPO-GNN in this experiment was roughly 12 hours, the majority of which is IPS solving time.

Table 3.4: Comparison of the mean and standard deviation of IPS convergence times and convergence rate of IPS between PPO GNN and the baseline on the 118-bus test case.

	Mean r_c	Standard Deviation r_c	Convergence %
PPO-GNN	**1.19 seconds**	**0.19 seconds**	**100.0%**
PyPower	1.70 seconds	0.39 seconds	99.4%
GNN	1.35 seconds	0.23 seconds	99.6%
PPO-MLP	1.39 seconds	0.28 seconds	97.9%

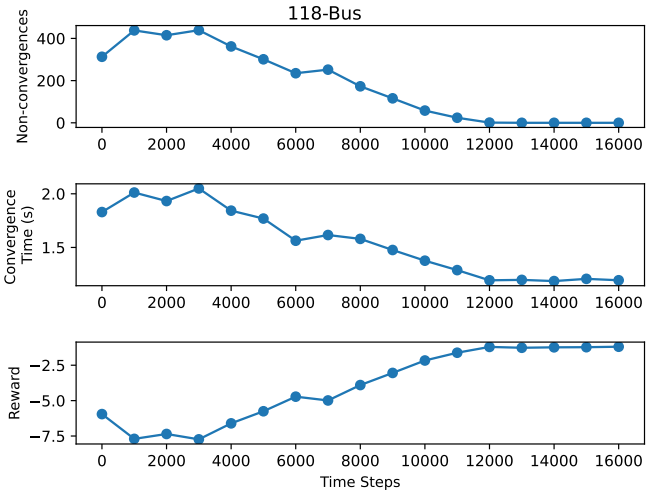

Figure 3.3: PPO-GNN training statistics for the 118-bus test case. The three scatter plots represent the number of non-convergences, average IPS convergence time, and average reward on the y-axis and the number of time steps on the x-axis.

3.6.3 300-bus

The IEEE 300-bus system contains 69 generators, 60 load tap changers, 304 transmission lines, and 195 loads. This is the largest and most complex of the power systems used in this study. As shown in Table 3.5, PPO-GNN outperforms all baselines but only with a marginal improvement over the PyPower ACOPF solver, which in previous experiments performed significantly worse than other models. This most likely indicates that other models performed poorly on this test case, rather than the PyPower ACOPF solver performing well. Figure 3.4 displays the training statistics for PPO-GNN on the 300-bus test case. In the early stages of training, the initial guesses provided resulted in non-convergence nearly 100% of the time but slowly decreased to nearly 0% after roughly 20,000 time steps of training. Similarly, convergence time decreased by a factor of over three from nearly 11 seconds to under three seconds from the start of training to the end of training. This indicates that the 300-bus test case is significantly more sensitive to the quality of an initial guess compared to smaller test cases, and it is likely that this trend will continue if experiments were to be conducted on larger, more complex systems. Training time for PPO-GNN on the 300-bus test case exceeded training time on other test cases by a substantial margin, requiring nearly 30 hours to fully train. Nonetheless, when fully trained, initial guess computation only takes about 30 milliseconds.

Table 3.5: Comparison of the mean and standard deviation of IPS convergence times and convergence rate of IPS between PPO GNN and the baseline on the 300-bus test case.

	Mean r_c	Standard Deviation r_c	Convergence %
PPO-GNN	**2.79 seconds**	**0.71 seconds**	**99.5%**
PyPower	2.98 seconds	0.84 seconds	99.3%
GNN	3.35 seconds	0.91 seconds	98.1%
PPO-MLP	3.43 seconds	0.87 seconds	95.9%

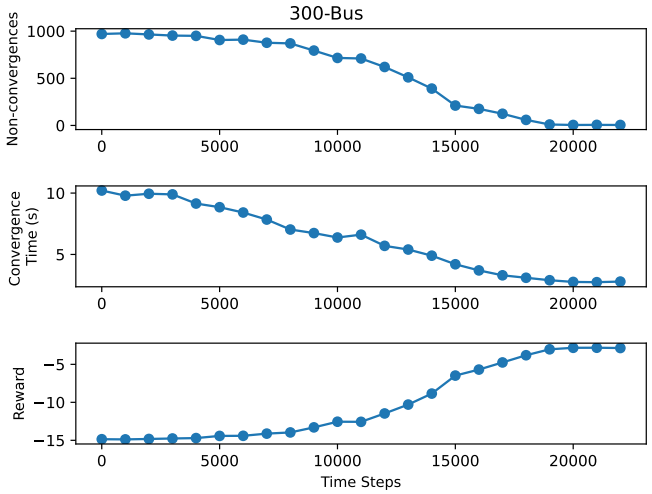

Figure 3.4: PPO-GNN training statistics for the 300-bus test case. The three scatter plots represent the number of non-convergences, average IPS convergence time, and average reward on the y-axis and the number of time steps on the x-axis.

3.6.4 Supplementary Experimentation

Additional experiments are conducted by sampling a wider range of initial guesses to observe how an initial guess may affect the quality of the resulting IPS convergence point in terms of cost, the ability to converge, and convergence time. This experiment also aims to understand if the reward function presented in (16) may cause the IPS to converge quickly but to a suboptimal solution. A uniform distribution is sampled within the system's limits of 2,000 unique initial guesses; then the resulting solution's costs and the IPS's convergence times are recorded, given these initial guesses, if it could converge. This experiment was conducted using the same random state on each test case. Results of this experiment are presented in Table 3.6. In every case, the IPS converged to the same point regardless of the initial guess, if it could converge. Still, convergence rates in this experiment were far lower than in previous experiments as initial guesses were randomly selected, indicating that IPS may be more sensitive to the quality of an initial guess than what was suggested in our previous experiments; the same can be said for IPS computation time. These results also suggest that the PPO-GNN can provide an initial guess that produces a fast solution without compromising the quality of the resulting ACOPF solution.

	Convergence %	Min. Cost $	PPO-GNN Cost $	Mean Convergence Time	PPO-GNN Convergence Time
14-Bus	76.7%	2.03E+07	2.03E+07	0.82 seconds	0.49 seconds
118-Bus	93.8%	3.63E+08	3.63E+08	1.50 seconds	1.15 seconds
300-Bus	34.2%	2.38E+09	2.38E+09	5.10 seconds	3.02 seconds

Table 3.6: Convergence rate, convergence time, and resulting cost of IPS given a random uniform distribution of initial guesses.

3.7 Conclusions

In this work, I proposed a novel framework for attaining quick, optimal solutions to ACOPF using PPO and GNN to compute initial guesses for an IPS. These initial guesses were assessed by whether or not the IPS could converge, given that initial guess and how quickly it could converge to the optimal solution. This resulted in a trained GNN that produces IPS initial guesses that minimize the number of non-convergences and significantly decrease IPS convergence time compared to three baselines: PyPower's ACOPF solver, initial guesses computed by a GNN that is trained to calculate ACOPF solutions directly, and an MLP trained with PPO using the same reward function as our proposed framework. This was demonstrated on three different IEEE test environments: the 14-bus test case, the 118-bus test case, and the 300-bus test case. In the future, I look to explore the use of this framework on much larger power system test cases to better understand the scalability of this solution, as well as instances of topology changes, such as line outages. Additionally, this work did not include a comprehensive assessment of alternative GNN architectures or different reinforcement learning algorithms, nor did I explore a wide variety of hyperparameter settings. With that, the solution presented in this chapter can be improved significantly, and I hope that this can provide a new direction for ACOPF research.

A key consideration for practical deployment is the computational cost associated with power system simulation. In this work, the proposed method is trained on benchmark systems where the cost of evaluating a single ACOPF instance is manageable. However, in more realistic or large-scale settings, simulation times can increase substantially, particularly when considering uncertainty, or more complex operational constraints. Since reinforcement learning methods require a large number of environment interactions, this can quickly become a limiting factor.

As problem complexity increases, not only does the cost of a single simulation step grow, but the number of samples required to learn effective policies also increases. In such settings, the approach proposed in this chapter may become computationally infeasible. Addressing this challenge will likely require the development of more sample-efficient reinforcement learning methods, such as model-based approaches or techniques that reduce the reliance on expensive simulation data. Improving sample efficiency is therefore a key step toward extending this framework to more realistic and large-scale power system applications.

This chapter focused on a particular power systems optimisation problem in which each time step is independent and therefore does not carry temporal context. However, many real-world systems, especially in power systems applications, exhibit strong temporal dependencies across time, where future system behavior depends not only on the current

state but also on historical observations and interactions between variables. Accurately modeling these temporal and cross-variable relationships is therefore vital for many forecasting and control tasks. The following chapter consequently shifts focus from optimisation to multivariate time series forecasting, introducing a Transformer-based framework designed to capture spatio-temporal dependencies in sequential data.

Chapter 4

Spatio-Temporal Transformer for MTS Forecasting

4.1 Introduction

Having shown in Chapter 3 that deep reinforcement learning combined with graph-based neural architectures can accelerate optimisation in power system environments, I now shift focus toward another fundamental challenge in modern machine learning: modeling temporal dependencies in multivariate sequential data. Many real-world systems evolve over time and generate large collections of interdependent variables whose behavior must be forecast accurately for effective monitoring, planning, and control. Examples include electricity demand and renewable generation in power systems, financial market activity, traffic flow, and weather patterns.

Many of these problems can naturally be framed as multivariate time series forecasting tasks. A multivariate time series (MTS) is a group of time-dependent variables, where each variable is represented as a sequence of historical data indexed in chronological order; each element in the sequence can exhibit dependencies on its historical values and other interlinked variables ([Silvestrini and Veredas, 2008]). For example, increased precipitation may be linked to decreased car traffic volume on a motorway. Multivariate time series data are prevalent across several domains, such as energy, finance, transportation, and healthcare ([Lu et al., 2022], [Patel et al., 2022], [Lee et al., 2021], [Merdjanovska and Rashkovska, 2022]), calling attention to the need for research and advancement of MTS forecasting. MTS forecasting can be defined as the prediction of a future value of a target variable based on historical values of that variable and other related variables ([Box et al., 2008]). Despite its importance, research on MTS forecasting has historically received comparatively less attention than univariate time series analysis, and many existing approaches fail to fully exploit the spatio-temporal dependencies inherent in such data [Lim and Zohren, 2021, Lara-Benítez et al., 2021]. In particular, accurately modeling both temporal evolution and interactions between variables remains a major challenge in large-scale real-world systems.

To allow for further exploration of application-based problems in which multivariate time series are a focal component, this chapter aims to address the gap in accurate and efficient MTS forecasting through the development of a Transformer-based framework designed specifically for spatio-temporal sequence modeling.

The ideal model for MTS forecasting should be able to fully exploit latent spatio-temporal dependencies to extract as much relevant information from the data as possible. Models developed for univariate time series can often be adapted slightly for multivariate time series and still achieve notable accuracy without a means for detecting spatial dependencies. Still many of the top-performing models for multivariate time series utilize mechanisms to identify both spatial and temporal dependencies ([Ruiz et al., 2021],[Wen et al., 2022]). The current state-of-the-art in time series is generally composed of neural networks that employ advanced methods that can capture these relationships, such as recurrent units, convolution, attention, graph learning, and many others. Before the rise of these deep learning methods, more rudimentary statistical analysis was commonly used for time series forecasting, such as autoregressive integrated moving average ([Box and Pierce, 1970]).

Although most commonly associated with computer vision applications, Convolutional

architectures have also become widely used in time series research. Their ability to detect dependencies within image pixels is easily transferable to time series, demonstrated in InceptionTime ([Fawaz et al., 2020]) and XceptionTime ([Rahimian et al., 2019]). InceptionTime is an ensemble of deep convolutional networks designed for univariate time series classification. XceptionTime utilizes depthwise separable convolutions and a novel normalization technique to operate on multivariate time series data but was only used for hand gesture classification. A non-deep convolutional model, Rocket ([Dempster et al., 2020]), currently scores high accuracy on many publicly available datasets by combining a novel method of random convolutional kernels and a linear classifier, outperforming or matching other time series models, including InceptionTime, across 85 University of California, Riverside (UCR) datasets for univariate time series classification. MiniRocket ([Dempster et al., 2021]), the successor to Rocket, utilizes very similar techniques and can produce comparable results to Rocket across 109 UCR datasets — univariate and multivariate time series classification.

Recurrent Neural Networks, namely long short-term memory (LSTM), have long been a staple in time series analysis. They utilize a hidden state to recall information from prior inputs to influence the current and future inputs and outputs, granting the ability to detect temporal behavior. LSTM-FCN ([Karim et al., 2018]) combines long short-term memory and a fully convolutional network to achieve improved accuracy over earlier models, such as ResNet ([He et al., 2016]) and COTE ([Bagnall et al., 2015]). ALSTM-FCN, an extension of LSTM-FCN, gains minor accuracy improvement with an added attention module. Initially designed for univariate time series, these models were later augmented for use in multivariate time series classification ([Karim et al., 2019]).

Despite their effectiveness, many of the aforementioned models are outclassed by ensemble models HIVE-COTE ([Lines et al., 2018]) and TS-CHEIF ([Shifaz et al., 2020]), who perform time series classification by combining various sophisticated classifiers. HIVE-COTE utilizes an ensemble of phase-independent shapelets, bag-of-words-based dictionaries, and phase-dependent intervals, while TS-CHEIF utilizes an ensemble of tree classifiers. Although these methods are powerful, they suffer from extremely high computational complexity, coupled with the inability to exploit GPU hardware well. Additionally, these models were built for univariate time series classification but did not consider multivariate time series or regression. HIVE-COTE 2.0 ([Middlehurst et al., 2021]), introducing Rocket classifiers, and novel dictionary and forest classifiers, can perform multivariate time series classification, but, similar to its predecessor, it cannot be used for regression and suffers from computational complexity issues.

Graph neural networks have become increasingly popular in recent years as they are effective in analyzing data with graph-like structures. These networks typically require the data to have a defined set of vertices, edges, and adjacency matrix.MTGNN ([Wu et al., 2020b]) introduces a novel implementation of a graph learning layer to use a graph neural network for multivariate time series. The graph learning layer allows the model to connect elements with strong relationships, allowing it to uncover spatio-temoral dependencies without needing a preexisting graph structure. MTGNN addresses single-step and multi-step forecasting across four multivariate time series datasets. MTPool ([Duan et al., 2022]) introduces a novel graph-pooling framework to hierarchically aggregate node information, addressing a significant limitation of graph neural networks, but is only evaluated on multivariate time series classification.

Similarly to the graph neural network, the Transformer ([Vaswani et al., 2017]), an attention-based neural network, has also increased in popularity since its inception. While initially developed for natural language translation, the Transformer is universally powerful across many sequence learning tasks, including time series ([Qi et al., 2021], [Li et al., 2019], [Zhou et al., 2021], [Wu et al., 2020a], [Wu et al., 2021], [Zhou et al., 2022a], [Liu et al., 2022d]). While the structure of the Transformer's attention mechanism is invariant to a sequence's order, positional embeddings are used to encode position information into each element in the sequence, informing attention about the sequence's order, thus allowing the Transformer to be effective for sequence learning.

Many Transformer-based models have been developed for multivariate time series applications. BAT ([Lei et al., 2022]) utilizes a Transformer-based framework for speech emotion recognition. Time Series Transformer (TST) ([Zerveas et al., 2021]) performs both multivariate time series classification and forecasting across 17 datasets using a Transformer encoder-based architecture, achieving higher accuracy than Rocket on a majority of studied datasets. TST used absolute position embeddings but did not specify an explicit means for capturing the data's spatial components. Spacetimeformer ([Grigsby et al., 2021]) couples absolute position embeddings with variable embeddings, informing the attention mechanism about both position information and variable information of each element, allowing it to capture spatio-temporal dependencies. This model achieves high accuracy across four multivariate time series datasets, outperforming MTGNN. ([Chen et al., 2022b]) combines a Transformer with a graph learning layer and graph convolution to perform anomaly detection in an Internet of Things system. The Gated Transformer Network ([Liu et al., 2021a]) utilizes two transformer encoders for multivariate time series classification: one encoder that attends to time steps and one encoder that attends to variables across a single time step. Transformers have also been widely adopted in video analysis tasks, which have similar spatio-temporal properties to MTS. Learning videos using transformers ([Neimark et al., 2021], [Ye and Bilodeau, 2023], [Qu et al., 2023]) demonstrates the ability of transformers to be effective across a wide range of tasks that require spatio-temporal learning.

Notwithstanding the success of recent Transformer-based models for multivariate time series, many did not utilize a structure that could fully exploit the spatio-temporal nature of a multivariate time series. Inspired by the Transformer, I propose Spatio-temporal Transformer with Relative Embeddings (STTRE). This model utilizes three different modules to learn spatial, temporal, and spatio-temporal dependencies by re-purposing relative positional representations to exploit these dependencies. STTRE is the first Transformer-based model for multivariate time series to introduce relative positional representations. Relative positional representations ([Shaw et al., 2018]), which I will refer to as relative embeddings for the remainder of this chapter, were introduced to replace the absolute positional embeddings in the Transformer — for natural language processing (NLP); this was a minor improvement. Still, it could be far more advantageous in time series applications. Using only absolute position to describe an element's position in a sequence can be misleading for many types of sequences. For time series, where the sequence is tied to observable time such as time of day, or day of the week, it can be arbitrary and misinformative to denote any time step in a sequence with a specific integer position, as that position could represent dissimilar times and/or days in different sequences. For example, position i could represent 00:00 and 06:00 in two different se-

quences, but they would be given the same position embedding. A visualization for this problem is provided in Figure 1. Many Transformer-based models use absolute positional embeddings as their only means of informing the attention mechanism about position information, but it is flawed in time series applications.

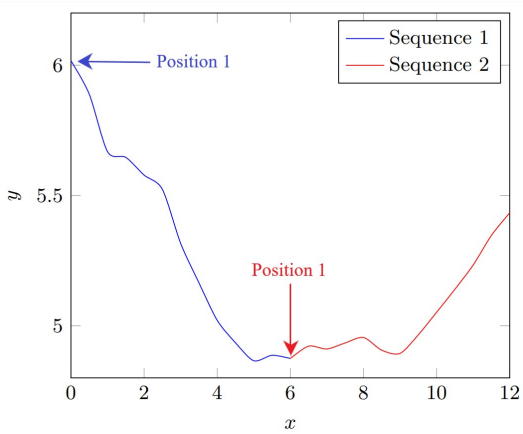

Figure 4.1: Two sequence segments from a toy dataset. The x-axis denotes the time of day in hours, while the y-axis denotes the value of a toy variable. Position 1 in sequence 1 represents the time 0:00, while position 1 in sequence 2 represents the time 6:00. While dissimilar, these positions will be given the same absolute position embedding.

The contributions of this work are as follows:

1. I create a novel three-module Transformer-based architecture to extract and isolate temporal, spatial, and spatio-temporal dependencies.
2. I introduce a novel implementation of relative embeddings to extract spatial and temporal dependencies in a multivariate time series. I also showcase a method of structuring multi-head attention to exploit my relative embedding implementation.
3. I implement the first Transformer-based model for multivariate time series to utilize relative embeddings.
4. I demonstrate that STTRE achieves the best accuracy across six experiments compared to other state-of-the-art multivariate time series models.

These contributions primarily focus on augmentations to the Transformer encoder, allowing the model to uncover latent relationships between elements more effectively than other Transformer-based implementations, granting a significant advantage for multivariate time series forecasting. I apply STTRE to perform forecasting across five multivariate time series datasets, covering energy consumption, traffic, finance, and air pollution — studying STTRE's accuracy on each. I compare performance with five baseline models, including top-performing models for multivariate time series, and demonstrate that STTRE outperforms all using various standard metrics. I also analyze the contribution of each component of the model by performing ablation studies, which adds to the interpretability of the results.

The rest of the chapter is structured as follows: Section 2 will cover background and related work, outlining details of the Transformer and its primary components; Section 3 will detail the proposed model, outlining the three core modules of the architecture and their characteristics; Section 4 will detail the experimental set-up and the datasets used, present the results, and analyze the performance of STTRE against the state-of-the-art; I shall conclude with a discussion and outline of future work in Section 5.

4.2 Background

In this section, I review background material and related work that supports STTRE. This includes a brief outline of the Transformer, embeddings, multi-head attention, and normalization.

4.3 Proposed Methodology

In this section, I address the problem of multivariate time series forecasting, where the objective is to predict a target variable using multiple correlated input variables (covariates). Let

$$X = \{x_1, x_2, \cdots, x_m\}$$

define a multivariate time series consisting of m variables, where each variable

$$x_i = \{x_{i,1}, x_{i,2}, \cdots, x_{i,l}\}$$

contains l historical observations, and $x_{i,j} \in \mathbb{R}$ denotes the value of the ith variable at time step j.

Given the historical multivariate observations X, the objective is to predict the next value of a target time series:

$$Y = y_{l+1},$$

where $y_{l+1} \in \mathbb{R}$ denotes the target variable at the next time step. In this formulation, the variables in X act as covariates used to forecast the target value Y.

4.3.1 Model

STTRE[1] incorporates three modules: the temporal module, the spatial module, and the spatio-temporal module. Each module is designed to capture different latent dependencies within a multivariate time series. Each module will accept a flattened and embedded multivariate time series segment $X_e \in \mathbb{R}^{lm \times d}$. The inputs are flattened into a lm length vector, and then each element is cast into the embedding dimension via linear transformation. Absolute position embeddings are still used in conjunction with relative

[1] https://github.com/AzadDeihim/STTRE

embeddings, as they offer a minor improvement in performance at a minimal computational cost — these are summed with element embeddings to produce X_e. In the spatial module, the absolute position embeddings are replaced with variable embeddings, which will embed information regarding the variable of an element rather than embedding information regarding the position of an element. For the position and variable embeddings, fully learnable positional embeddings are used.

Each module utilizes a structure similar to the Transformer's encoder but with augmentations that I hypothesize will improve its ability to perform its delegated task. I also adopt the encoder's layering mechanism, setting the number of layers to three.

The output of each module, $X'_e \in \mathbb{R}^{lm \times d}$ is concatenated, producing a new matrix $X_{st} \in \mathbb{R}^{3lm \times d}$, a spatio-temporal encoded representation of X_e. A feed-forward layer consolidates the embedding dimension such that X_{st} is reduced to a vector of length $3lm$. A linear regression layer reduces this vector into a single value, representing the final prediction, Y. A diagram of STTRE's architecture can be found in Figure 4.2.

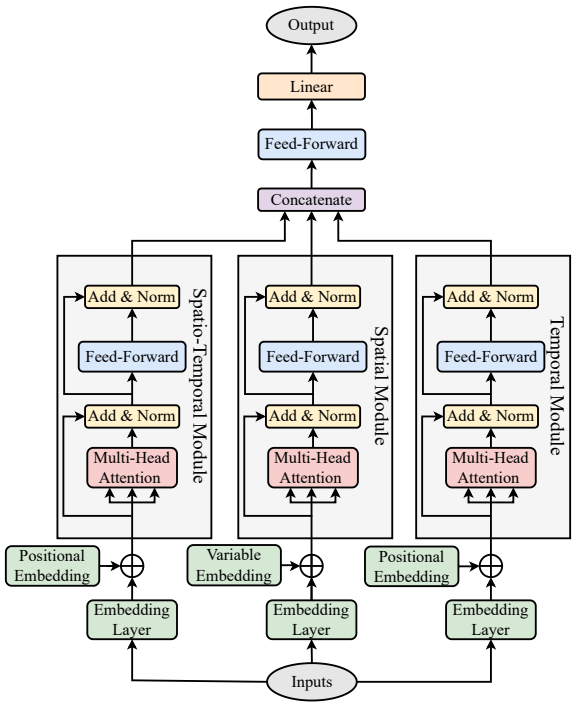

Figure 4.2: Diagram of STTRE Model.

Learned weights in the architecture were optimized using a mean squared error loss function:

$$\textbf{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \tag{4.1}$$

where n is the batch size, Y is the observed target value, and $\hat{Y}$ is the predicted target value.

In most cases, STTRE requires relatively small values of d for optimal performance,

ranging from 8 to 32. Many Transformer implementations require significantly larger values of d for optimal results; for instance, Time Series Transformer, a Transformer-based framework for multivariate time series ([Zerveas et al., 2021]), uses embedding sizes ranging from 256 to 512. I found that using a smaller embedding size increases accuracy significantly while also greatly decreasing computation time and memory requirements. I demonstrate the trade-off between accuracy and embedding size in 4.4.5.

Multi-Head Attention Structure

While the architecture of each module appears the same, the structure of the multi-head attention in each differs. The three core modules, temporal, spatial, and spatio-temporal, are designed to learn information and detect dependencies related to their named functions; thus, each module's multi-head attention structure is designed to improve its ability to perform its assigned function.

The temporal module is designed to detect strictly temporal dependencies while minimizing its ability to capture spatial dependencies. In this module, multi-head attention will have exactly m heads, one head to attend to each variable. Also, X_e will be divided along the m dimension rather than the d dimension, resulting in a (l, d) space for each head so that heads cannot attend to variables other than the one allocated to it. Lastly, weight matrices will have separate weights for each head. There are no learned weights in the temporal module that are shared across multiple variables, so each weight will only be capable of learning information regarding the variable it was delegated to.

The spatial module behaves similarly to the temporal module but transposed — X must be transposed before transforming into X_e. In this module, multi-head attention will use exactly l heads, one head to attend to each time step. X_e is divided along the l dimension, resulting in a (m, d) space for each head, so each head cannot attend to time steps other than the one allocated. Weight matrices will have separate weights for each head — weights will be capable of learning information across variables but only within their assigned time step.

Multi-head attention in the spatio-temporal module will be constructed differently than in the spatial and temporal modules — it is more similar to that of a standard implementation. It will have four heads, with each attending to $\frac{1}{h}$ of the embedding space, resulting in a $(lm, \frac{d}{h})$ dimensional space for each head. Unlike the other modules, the heads in the multi-head attention will be granted access to elements across different variables and timesteps but are confined to their assigned portion of the embedding space. Weight matrices will share weights across heads. A visual representation of the multi-head attention's functionality for each module is provided in Figure 4.3.

4.3.2 Relative Embeddings

Relative embeddings are employed to aid each module in constructing an encoded spatio-temporal representation of X_e. To obtain relative embeddings, I must first generate a matrix of learnable relative embedding weights E_r. The operation shown in (4.3) is

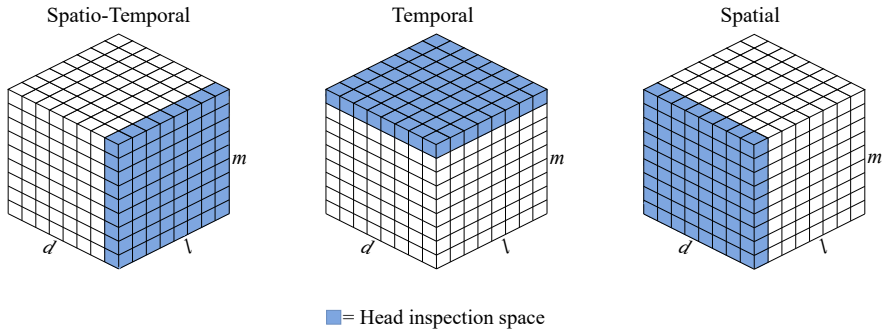

Figure 4.3: Head inspection space for each module. This diagram is meant to visualize the dimensions each module attends to. For simplicity, only the inspection space for the first head is shown.

performed using Q, the queries, and E_r to obtain S, a square matrix containing an entry for all pairs of elements in X, denoting a relationship between the elements' locations in X. Next, the scaled dot-product attention function is augmented to factor relative embedding information:

$$\text{Attention}(Q, K, V) = \text{Softmax}(\frac{QK^T + S}{\sqrt{d/h}})V, \tag{4.2}$$

$$S = \text{skew}(QE_r^T). \tag{4.3}$$

An upper left triangle mask is applied to QE_r^T before the skewing function. The skew function is crucial in this calculation as S must match up with the indexing in QK^T. Otherwise, the addition calculation in (4.2) will add relative embeddings to the incorrect indices. Each (i, j) entry in QK^T contains the dot product of the query for element i in X and key for element j in X, making QK^T absolute-by-absolute indexed. However, When QE_r^T is computed, each (i, r) contains the dot product of the query at position i and the relative embedding r between elements i and j in X, making QE_r^T absolute-by-relative indexed. The skew function's purpose is to shift columns in QE_r^T to make the indexing absolute-by-absolute. ([Huang et al., 2018]). Algorithm 1 describes the skewing function, defined in the context of the temporal module. Dimensionality will vary in other modules, but the algorithm's structure will remain unchanged. Figure 4.4 visualizes how the skewing function operates.

Algorithm 1: Skew

Input: $S_0 \in \mathbb{R}^{l \times l}$, the resultant matrix of QE_r^T with an upper left triangle mask.
Output: $S \in \mathbb{R}^{l \times l}$, the skewed representation of S_0.

$S_p \leftarrow \text{pad}(S_0)$ {Pad column of zeros on left}
$S_r \leftarrow \text{reshape}(S_p)$ {Such that upper left triangle mask is now an upper right triangle mask}
$S \leftarrow \text{slice}(S_r)$ {Remove first row}

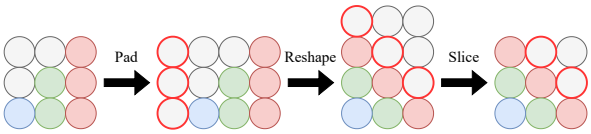

Figure 4.4: Diagram of skewing function — grey circles represent masked values. Firstly, I pad a column vector before the leftmost column. Next, I reshape the matrix such that the upper triangle mask is now on the right. Lastly, I remove the first row to produce the desired S.

In each module, relative embeddings will capture pairwise relationships between elements within their respective head inspection spaces. In the temporal module, the relative embedding weight matrix $E_t \in \mathbb{R}^{h \times l \times d}$, have a separate (l, d) dimensional weight matrix for each head as the relative embeddings of each head should only be able to acknowledge pairs within its assigned variable, and not be able to acknowledge pairs across variables. The resulting matrix $S_t \in \mathbb{R}^{h \times l \times l}$ will be a matrix of pairwise temporal relationships.

In the spatial module, $E_s \in \mathbb{R}^{h \times m \times d}$ has a separate (m, d) dimensional weight matrix for each head, as the relative embeddings of each head should only be able to acknowledge pairs within its assigned time step, and not be able to acknowledge pairs across time steps. Thus $S_s \in \mathbb{R}^{h \times m \times m}$ will be a matrix of pairwise spatial relationships.

In the spatio-temporal module, $E_{st} \in \mathbb{R}^{lm \times \frac{d}{h}}$ has shared weights across heads. This is done to reduce memory requirements but cannot be done in other modules as it may obstruct their ability to perform their delegated task. Relative embeddings in this module will be capable of recognizing pairs across variables and time steps, thus $S_{st} \in \mathbb{R}^{lm \times lm}$ will be a matrix of pairwise spatio-temporal relationships.

Figure 4.5 provides a visual representation of each module's inspection space of relative embeddings. Algorithm 2 describes multi-head attention with relative embeddings for the temporal module. The algorithm's structure is similar for the other modules, but the dimensions must be adjusted accordingly.

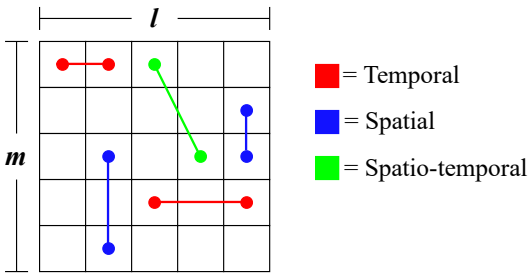

Figure 4.5: Visual representation of the span of relative embeddings in each module. Note that not all possible pairwise connections are represented in this figure.

Algorithm 2: Temporal Multi-head Attention With Relative Embeddings

Input: $X_e \in \mathbb{R}^{lm \times d}$, the flattened and embedded input multivariate time series; h, the number of heads.
Output: $z \in \mathbb{R}^{lm \times d}$, the new attention-weighted temporal representation of X_e
Parameters: Linear transformation weights and biases for the values, keys, and queries,

$W_V \in \mathbb{R}^{d \times d}, \quad b_V \in \mathbb{R}^d$
$W_K \in \mathbb{R}^{d \times d}, \quad b_K \in \mathbb{R}^d$
$W_Q \in \mathbb{R}^{d \times d}, \quad b_Q \in \mathbb{R}^d$

Relative embeddings weights, $E_r \in \mathbb{R}^{h \times l \times d}$

$V \leftarrow W_V X_e + b_V$
$K \leftarrow W_K X_e + b_K$
$Q \leftarrow W_Q X_e + b_Q$
$S_0 \leftarrow \text{mask}(QE_r^T)$ {Upper left triangle mask}
$S \leftarrow \text{skew}(S)$
$z \leftarrow \text{Softmax}(\frac{QK^T+S}{\sqrt{d/h}})V$

4.4 Experiments

The results shown in this section are produced by training models on a predefined set of training data and evaluating them on a predefined set of testing data. The training data and testing data will be the same for all models. All models are trained until convergence. The number of epochs until convergence varied between models and datasets. Experiments are conducted using Python 3.7 on an NVIDIA A100 40GB GPU. For each epoch, the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE)[2], and mean absolute error (MAE) of the predictions on testing data is recorded. After each epoch, the average of each metric across the five most recent epochs is also recorded — the lowest average MSE will be used to determine which epoch to extract results from. The results will be presented as an average across five epochs.

4.4.1 Data

Five publicly accessible datasets are used for this study. Datasets are primarily gathered from the University of California, Irvine (UCI) Machine Learning Repository ([Dua and Graff, 2017]), except for one aggregated from Yahoo! Finance.

- **Metro Interstate Traffic Volume**: a dataset from the UCI repository used to predict car traffic volume on an interstate highway.

[2]MAPE is only considered for datasets where the zero values do not exist in the target variable, as this will produce divide-by-zero errors.

Dataset	Sampling Rate	Total Samples	Sequence Length	Variables
MetroInterstate	1 hour	48205	24	17
Uber Stock	1 day	726	60	5
Appliances Energy	10 minutes	19736	144	26
BeijingPM2.5	1 hour	15901	24	6
Istanbul Stock	1 day	536	40	8

Table 4.1: Dataset characteristics according to sampling rates, number of samples, sequence lengths, and number of variables. The heterogeneous dataset selection is designed to provide STTRE with an extensive assessment.

- **Uber Stock**: a dataset aggregated from publicly available Yahoo! Finance data ([Yahoo!Finance, 2022]). Used to predict the stock price of Uber.
- **Appliances Energy**: a dataset from the UCI repository used to predict household energy usage. This dataset contains two target variables: energy consumption of appliances and energy consumption of lights — these target variables will be evaluated separately, denoted by 1 and 2, respectively.
- **BeijingPM2.5**: a dataset from the UCI repository used to predict air pollution in Beijing.
- **Istanbul Stock Exchange**: a dataset from the UCI repository used to predict Istanbul stock exchange returns.

For additional context, the root mean square (RMS) magnitude of the daily price changes was approximately 1.32 for the Uber dataset and 0.023 for the Istanbul Stock Exchange dataset. These values provide a reference scale for interpreting the forecasting errors relative to the typical day-to-day variability of each series.

Table 4.1 describes the characteristics of each examined dataset. While the datasets cover various domains, they also provide diverse characteristics, allowing for a comprehensive examination. I used a 50%-50% split of training and testing data for each dataset. 20% of the training data was used as validation data; validation data is used strictly to tune hyperparameters for the model(s). The datasets are not randomized prior to the split. Chronologically, all data in the testing set occurs after all data in the training set. This is a more practical approach to time series problems, as real-world applications will always exhibit this characteristic. Also, this mitigates the presence of high degrees of autocorrelation between sequences in the training set and sequences in the test set since chronologically neighboring sequences will have significant overlap and will generally be easy targets to predict accurately.

Multivariate time series data can often exhibit different behavior based on long-term trends, i.e., seasonal trends. The train and test split used in this study allows the models to be evaluated on a more diverse set of data, encompassing various monthly or seasonal trends. If the test set is too small, I may only find that models are trained on data from various months or seasons but are only evaluated on data from one season, thus providing only a limited assessment of the model(s).

	RMSE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	**895.6**	**2.947**	**90.74**	**6.474**	**36.02**	**0.013**
XceptionTime	2011.6	4.342	108.7	8.473	40.16	0.016
TST	936.7	5.926	108.1	7.937	38.91	0.036
MiniRocket	1206.1	6.458	103.2	8.639	47.85	0.047
LSTM	974.4	4.678	109.4	9.244	41.26	0.016
LSTM-FCN	1100.8	5.190	103.8	8.213	47.61	0.018

	MAE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	**556.7**	**2.216**	**48.65**	**3.649**	**20.96**	**0.008**
XceptionTime	1769.5	2.758	58.64	4.970	24.25	0.012
TST	569.6	3.714	62.30	4.671	24.14	0.029
MiniRocket	888.5	4.531	55.43	5.884	32.10	0.036
LSTM	617.3	3.088	60.10	6.072	25.65	0.012
LSTM-FCN	802.2	3.236	62.95	5.301	31.34	0.014

	MAPE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	-	**0.069**	**0.513**	-	-	**0.731**
XceptionTime	-	0.085	0.672	-	-	0.973
TST	-	0.115	0.776	-	-	6.546
MiniRocket	-	0.148	0.641	-	-	8.623
LSTM	-	0.094	0.708	-	-	1.438
LSTM-FCN	-	0.103	0.845	-	-	2.276

Table 4.2: Results of each experiment. Bold values indicate the best score for each category, and underlined values represent the second-best score for that category.

4.4.2 Baseline Models

STTRE's performance is compared with a diverse selection of 5 baseline models. These models are primarily selected from recently published literature which include multivariate time series analysis in their study and hence are representative of the state-of-the-art.

- **XceptionTime**: A convolutional neural network for multivariate time series ([Rahimian et al., 2019]).
- **TST**: A transformer-based neural network for multivariate time series ([Zerveas et al., 2021]).
- **MiniRocket**: Random convolutional kernels with linear regression layer for prediction ([Dempster et al., 2021]).
- **LSTM**: Long short-term memory ([Hochreiter and Schmidhuber, 1997]).
- **LSTM-FCN**: Long short-term memory with a fully convolutional network ([Karim et al., 2018]).

4.4.3 Results

I conducted six experiments using the five previously outlined datasets. Table 4.2 shows the RMSE, MAE, and MAPE scores earned by each model on each dataset. The winner in each experiment is denoted in bold font, while an underline denotes the runner-up. I observe that STTRE outperforms all baselines across all datasets, achieving an average rank of 1. The second-best-performing model, XceptionTime, earns an average rank of 3.4, directly followed by TST. Compared with XceptionTime, STTRE achieves a median 27% improvement in RMSE, 23% improvement in MAE, and 23% improvement in MAPE.

With the comprehensive dataset selection considered in this study, I observe STTRE's ability to forecast multivariate time series in various environments, including datasets from different domains with vastly differing sizes, sampling rates, sequence lengths, and number of variables. STTRE's performance remained consistent across all experiments, demonstrating that it can excel across a wide range of datasets, regardless of their characteristics. On the contrary, while TST and XceptionTime performed well on many of the datasets, both models obtained significantly lower accuracy than other models on at least one dataset: TST ranked 5^{th} on the Istanbul stock exchange, and XceptionTime ranked 6^{th} on the Metro Interstate Traffic Volume. This suggests that these models are less robust and have a more narrow usability range than STTRE.

For each experiment, I compare STTRE's accuracy against the second-best performing model, XceptionTime, and the median accuracy of all baseline models, calculating the improvement in accuracy as a percentage. This is shown in Table 4.3. Although the margins are notable in all experiments, STTRE scores most decisively on Uber Stock and Istanbul Stock Exchange, two small financial datasets. While this could indicate that STTRE works best on small datasets, the more likely hypothesis is that the other models underperform on small datasets, as this is a complication for many deep learning models.

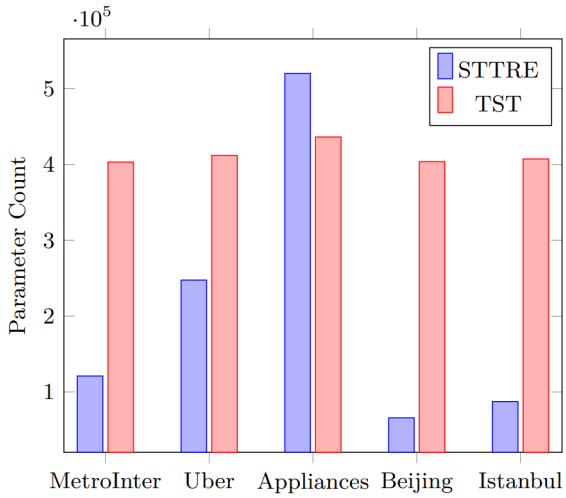

Figure 4.6: Parameter count of STTRE compared with parameter count of TST.

TST, the other Transformer-based model, is the runner-up in three of the six experiments. It utilizes a relatively simple architecture, incorporating one Transformer encoder rather than three and does not utilize relative embeddings. Still, the parameter count in TST outnumbers that of STTRE in most experiments. With a better allocation of learned

	Improvement	
Experiment	vs. XceptionTime	vs. median
MetroInterstate	3.3%	17.6%
Uber Stock	24.0%	33.4%
Appliances 1	14.7%	19.3%
Appliances 2	20.1%	25.6%
BeijingPM2.5	10.3%	13.7%
Istanbul Stock	23.7%	39.1%

Table 4.3: STTRE's improvement in accuracy over the runner-up, XceptionTime, and the median accuracy of all baseline models.

weights, STTRE can use fewer weights on average to yield state-of-the-art accuracy. Figure 8 provides a visual comparison of parameter counts of TST and STTRE. Information regarding hyperparameter settings, training, and convergence is provided in 4.6.

4.4.4 Ablation study

In this section, I conduct an ablation study to demonstrate the indispensability of key components in this architecture. The following models will be evaluated:

- **w/o RE**: STTRE without relative embeddings.
- **w/o STM**: STTRE without the spatio-temporal module.
- **w/o SM**: STTRE without spatial module.
- **w/o TM**: STTRE without the temporal module.

The ablation study is conducted using the same datasets which I examined previously. Table 4 contains the results of the ablation study, displaying the RMSE, MAE, and MAPE earned by each model on each dataset. The winner in each experiment is denoted in bold font, while an underline denotes the runner-up. The ablation study demonstrates that with all components and modules intact, the base model achieves the best results, earning an average rank of 1.3. I calculate the significance of each removed component across all datasets, given as a percentage, denoting the improvement of STTRE's accuracy compared to the other versions in the ablation study. In Table 4.5, I report the average improvement in accuracy granted by each component and the standard deviation of improvement in accuracy.

On average, w/o SM scores the lowest accuracy, which tends to remain consistent across all experiments. This suggests that the spatial module could be the STTRE's most influential component. Performance degradation w/o SM does not coincide with the number of variables in the datasets or any other dataset characteristic. This also appears to be the case for the other modules.

On Uber Stock and Istanbul Stock Exchange, STTRE performed relatively well without relative embeddings. I hypothesize that this is due to the small amount of data used

	RMSE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	**895.6**	**2.947**	**90.74**	**6.474**	**36.02**	**0.013**
w/o RE	903.7	3.033	93.54	6.873	37.33	**0.013**
w/o STM	919.6	4.022	93.23	6.727	36.42	0.016
w/o SM	927.5	4.939	94.98	6.922	37.22	0.016
w/o TM	933.2	4.898	91.80	7.328	36.14	0.014

	MAE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	**556.7**	**2.216**	48.65	**3.649**	**20.96**	**0.008**
w/o RE	563.8	2.222	52.64	3.981	21.46	**0.008**
w/o STM	578.5	2.621	51.05	3.843	21.29	0.012
w/o SM	570.6	3.412	51.59	4.022	21.20	0.011
w/o TM	577.4	3.371	**47.27**	4.193	21.01	0.009

	MAPE					
Model	MetroInter.	Uber Stock	Appliances 1	Appliances 2	BeijingPM2.5	Istanbul Stock
STTRE	-	**0.069**	0.513	-	-	**0.731**
w/o RE	-	0.071	0.625	-	-	0.838
w/o STM	-	0.079	0.579	-	-	1.237
w/o SM	-	0.101	0.576	-	-	1.189
w/o TM	-	0.097	**0.501**	-	-	0.826

Table 4.4: Ablation study — bold values indicate the best score for each category, and underlined values represent the second-best score for each category.

Component	Improvement	SD
Relative Embeddings	4.6%	4.9%
Spatio-Temporal Module	11.8%	11.5%
Spatial Module	15.4%	13.8%
Temporal Module	11.3%	13.0%

Table 4.5: Improvement in accuracy and standard deviation (SD) of accuracy improvement given by each component.

for training in these datasets, as both of these datasets are two orders of magnitude smaller than the other datasets used in this study. It is likely that relative embeddings require larger amounts of data for full efficacy and may be less advantageous on smaller datasets. I find that on the subset of datasets containing larger amounts of data, relative embeddings provide an average 5.6% improvement in accuracy.

In some cases, STTRE can still obtain high accuracy w/o TM, demonstrated on Appliances Energy 1, BeijingPM2.5, and Istanbul Stock Exchange. This could be due to redundancy in the spatio-temporal module. For example, if the temporal module is removed, STTRE will still have the means to undercover temporal dependencies via the spatio-temporal module. Still, it is important to note that since the heads in these modules attend to different dimensions, they will both hold different interpretations of the uncovered temporal dependencies. Alternatively, this could result from those datasets having weak temporal dependencies.

4.4.5 Embedding Size and Accuracy Trade-Off

I demonstrate the trade-off between embedding size and accuracy to validate the claim that smaller embedding sizes work better for STTRE. Figures 4.7 and 4.8 display the trade-off between RMSE and embedding size on the Uber Stock and BeijingPM2.5 datasets, respectively. These figures show that commonly used embedding sizes, which generally range between 256 and 512, do not work well for STTRE.

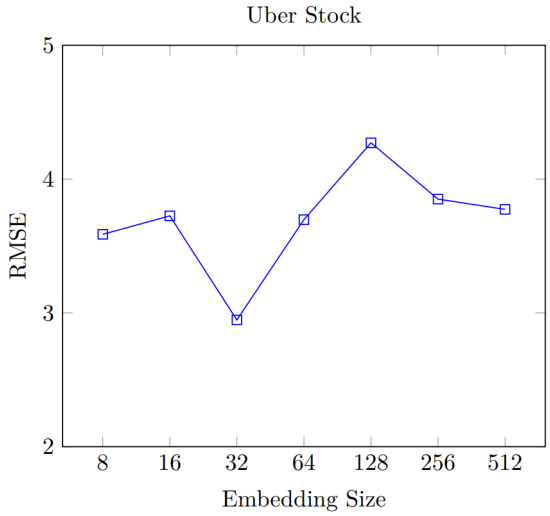

Figure 4.7: RMSE versus embedding size on Uber Stock dataset.

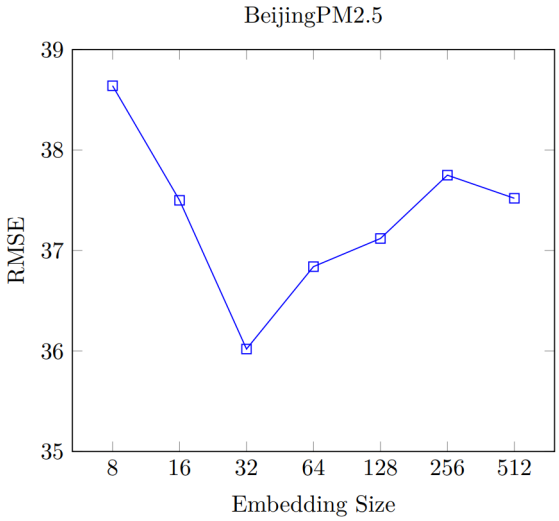

Figure 4.8: RMSE versus embedding size on BeijingPM2.5 dataset.

4.5 Conclusion

In this work, I proposed STTRE, a novel Transformer-based model for spatio-temporal learning of multivariate time series. STTRE addresses shortcomings of Transformer-

based models for multivariate time series by improving upon the encoder in a way that enhances recognition of spatio-temporal dependencies, namely the novel implementation of relative embeddings, coupled with a novel restructuring multi-head attention to fully exploit latent spatio-temporal dependencies in a multivariate time series. I evaluated STTRE on a comprehensive set of publicly available multivariate time series forecasting datasets encompassing an expansive range of characteristics. I demonstrate significantly improved performance over several state-of-the-art models, showing STTRE to be the best-performing model among the evaluated baselines on the selected benchmark datasets.

The innovations introduced in this study focused primarily on computing an encoded spatio-temporal representation of a multivariate time series. This was done via the three core modules in the architecture. The complementary task of computing a prediction given the encoded representation of the data was originally a task delegated to the decoder, which is absent in this work and replaced with feed-forward and linear layers. In the future, I aim to explore the use of alternative mechanisms that could aid in computing a prediction given the encoded representation of the data in lieu of the decoder. Additionally, although STTRE tends to be exceptionally memory efficient when compared to other Transformer-based models, due to the nature of relative embeddings, memory requirements can scale poorly as sequence lengths and the number of variables increase. In the future, I aim to find a remedy for this issue when faced with larger inputs.

More broadly, the ideas explored in this chapter highlight the importance of learning expressive representations that can fully exploit strong temporal and cross-variable dependencies where they are present. While this chapter focused on forecasting future values from historical multivariate observations, many real-world environments extend this idea, requiring sequential decision-making based on temporally evolving system dynamics. This, along with Chapter 3 naturally motivates the following chapter, which extends Transformer-based sequence modeling into the domain of multi-agent reinforcement learning through the development of a Transformer world model for sample-efficient learning.

4.6 Reproducibility

For reproducibility of the experiments, I outline hyperparameter settings and the number of epochs and total computation time for convergence for each experiment, shown in Table 4.6. I set dropout to 0.1 for regularization for all experiments and used the Adam optimizer. The weight matrices in the feed-forward layer of each module are (d, d) dimensional.

Dataset	Learning Rate	d	Batch Size	Epochs	Time
MetroInterstate	0.0001	32	256	1070	328
Uber Stock	0.0001	32	64	1320	29
Appliances Energy 1	0.0001	8	32	47	371
Appliances Energy 2	0.0001	8	32	51	396
BeijingPM2.5	0.0001	32	256	1063	86
Istanbul Stock	0.00005	32	268	485	5

Table 4.6: Hyperparameter settings for all experiments. This table includes settings for learning rate, embedding size, batch size, number of epochs until convergence, and total training time in minutes.

Chapter 5

Transformer World Model for Multi-Agent RL

5.1 Introduction

Building on the previous chapter, which demonstrated how Transformer architectures can be adapted for spatio-temporal forecasting in multivariate time series, I now turn to the challenge of sequential decision-making in multi-agent systems. Many problems in power systems and related domains involve multiple interacting agents, such as distributed generators or autonomous market participants. In these environments, each agent must reason not only about temporal dynamics but also about the actions and intentions of other agents. Multi-Agent Reinforcement Learning (MARL) has emerged as a natural framework for such settings, but it is often hindered by poor sample efficiency, instability, and difficulty in scaling to complex interactions.

These challenges become particularly severe in physical systems, where each interaction may require computationally expensive simulation. This was evident in Chapter 3, where a single simulation step could take several seconds to compute, making training over millions of interactions potentially infeasible. Similar limitations arise in load-frequency control (LFC), which is commonly studied using MARL, where simulation costs increase substantially with larger power system topologies. As system complexity increases, both the computational cost per interaction and the amount of data required for learning grow significantly, limiting the applicability of standard reinforcement learning methods in many real-world domains. As a result, many LFC studies employing MARL report experiments only on relatively small systems with only two or three generators [Rozada et al., 2020, Muduli et al., 2024]. This motivates the need for more sample-efficient architectures capable of learning accurate environment dynamics and coordinating multiple agents effectively. The Multi-Agent Transformer World Model proposed in this chapter is developed with this objective in mind.

Reinforcement learning (RL) is a framework in which agents learn to make decisions by interacting with an environment to maximize cumulative rewards, typically through trial and error. Deep RL (DRL) extends this by using deep neural networks to approximate policies and value functions, enabling agents to operate in high-dimensional or complex environments. Within DRL, model-free algorithms refer to methods that learn directly from experience without constructing an explicit model of the environments dynamics. These include popular approaches like Q-learning and policy gradients. Model-free DRL methods have led to breakthroughs in many DRL settings [Foerster et al., 2018, Lowe et al., 2017, Yu et al., 2022, Rashid et al., 2020, Wang et al., 2021, Son et al., 2019, Sunehag et al., 2018], but they often suffer from sample inefficiency, requiring millions of interactions to learn effective behaviors. This efficiency gap has limited the use of RL in real-world tasks where large-scale interaction is not feasible [Hernandez-Leal et al., 2020]. World models, however, offer a promising solution to this issue. In RL, a world model is a learned representation of the environment that allows agents to simulate and learn from predicted future outcomes without interacting with the real environment [Ding et al., 2024]. By enabling agents to "imagine" future trajectories given some contextual observations and actions, they can compute and learn from up to 16 imagined future steps per real sample. A world model typically uses neural networks to replicate environment dynamics like transitions and rewards. This capability dramatically improves sample efficiency, which can make DRL more attractive for domains where real-world sampling

of millions of experiences is impractical, such as robotics, healthcare, or autonomous driving systems.

While world models have demonstrated substantial improvements in sample efficiency for single-agent RL, their application to multi-agent environments remains largely unexplored due to the added complexity of modeling interactions among agents. Only a few existing frameworks [Xu et al., 2022, Zhang et al., 2024, Egorov and Shpilman, 2022, Zhang et al., 2025, Toledo, 2024] have attempted to bring world modeling into multi-agent RL (MARL), and although these models achieve promising results, they still fall short of matching the complexity needed to tackle environments on par with those addressed by their single-agent counterparts, such as vast 3D environments like Minecraft [Guss et al., 2019].

A major advantage of using world models in multi-agent systems is the ability to apply new dynamics to old experiences. Generally speaking, learning from older experiences in MARL has the potential to become harmful because they reflect outdated behaviors from other agents, but a world model with the capacity to track and update with each agents evolving behavior can reinterpret old experiences given new dynamics, making old transitions more relevant. Multiagent imagination rollouts can be modeled either centrally, where one model imagines a joint trajectory using all agents states and actions collectively, or independently, with one rollout per agent. Centralized imagination captures interagent dependencies, but the joint state grows with the number of agents and observations, leading to infeasibly large weight matrices and attention blocks whose parameter count and memory traffic scale at least quadratically with the number of agents. Small model errors also couple across agents, so the joint decoder can compute mutually inconsistent configurations, such as two or more agents occupying the same space in their own respective observations. Independent imagination scales linearly and keeps each agents state internally consistent, avoiding these impossible joint states; the cost being less explicit modeling of crossagent influence. While linearly scalable, purely local, independent imagination faces policy-induced nonstationarity: each agents transition model must track other agents whose policies evolve over time, so imagined state transitions can quickly become inaccurate if the world model is not able to keep up with other agents' evolving behaviors. Also, when using world models in multi-agent settings, accurately predicting long-horizon trajectories becomes increasingly difficult, as the interactions of multiple agents introduce additional sources of error that can compound and propagate throughout the entire imagination rollout. As a result, long horizon lengths that are feasible in single-agent settings often become infeasible in multi-agent environments.

To advance the capabilities of multi-agent world models, I introduce a Multi-Agent Transformer World Model (MATWM)[1], built upon STORM [Zhang et al., 2023], a transformer-based world model for single-agent systems. STORM consolidates several best practices from recent world models [Hafner et al., 2020, 2024, Robine et al., 2023], including transformer sequence modeling [Vaswani et al., 2017], KL balance with free bits [Kingma et al., 2016], percentile return normalization, two-hot symlog reward targets, and discrete latent representations, which is extended to the multi-agent setting. I select STORM as the foundational architecture due to its transformer-based sequence modeling, which offers advantages over recurrent approaches such as DreamerV3. In particular, transformers are

[1] github.com/azaddeihim/matwm

better suited for capturing long-range temporal dependencies, which are vital for modeling multi-agent settings where coordination depends on sequences of interdependent states and actions across agents. Additionally, transformer architectures enable more flexible representation learning and parallel computation, improving scalability as the number of agents increases. Also, STORM has been empirically shown to outperform DreamerV3 in single-agent settings, making it a strong foundation for extension to MARL.

The design uses independent, per-agent imagination with a shared world model, while the transition dynamics for a focal agent condition only on its own action, and teammates are treated as part of the stochastic environment. To recover coordination benefits without requiring joint trajectories or access to other agents private observations, I introduce a teammate-action predictor that supplies semi-centralized guidance to the per-agent actorcritic during training. I further stabilize imagined rollouts under partial observability and legality constraints via an action-mask predictor, and I mitigate nonstationarity with recency-prioritized replay.

The primary contributions are as follows:

1. I solve complex multi-agent benchmark, the StarCraft Multi-Agent Challenge (SMAC), with as few as 50K real environment interactions, which, to my knowledge, is the lowest sample allowance of any existing MARL algorithm.

2. A per-agent imagination scheme in which the world models transition does not take joint actions, paired with a teammate-action predictor that provides predicted teammate action probabilities to the policy/critic where they would have otherwise been unavailable.

3. An action-mask predictor that enforces action feasibility during imagined rollouts, improving stability in partially observable or constrained environments.

4. A replay sampling strategy that biases world-model updates toward recent trajectories to better track evolving policies of other agents in order to combat nonstationarity.

5. A cohesive training setup that ports effective single-agent components (KL balance with free bits, percentile normalization, symlog two-hot rewards, discrete latents) to multi-agent learning, while introducing new and improved formulations for these methods.

Additionally, both the teammate action and action mask predictors provide directly supervised auxiliary losses whose gradients propagate into the encoder/latent, enhancing the latent representation of the observation around coordination- and legality-relevant features. Also, while SMAC motivates the design, the action-mask predictor is broadly applicable; as legality constraints are ubiquitous in discrete multi-agent domains [Terry et al., 2021, Lanctot et al., 2019, Côté et al., 2018, Huang et al., 2021]. Even in environments that default to treating illegal actions as no-ops, which can waste exploration and destabilize credit assignment; manually replacing no-ops with a legal-action mask consistently reduces search space and improves sample efficiency [Wilson et al., 2024, Nazari et al., 2018, Zhao et al., 2024].

The remainder of this chapter is structured as follows. In Section 5.2, I review prior work on world models and their extension to MARL. Section 5.3 details the architecture and training process of MATWM, including my key design choices and novel components. Section 5.4 presents experimental results across SMAC, PettingZoo, and MeltingPot environments, demonstrating the effectiveness and efficiency of my approach and analyzing the contribution of individual components via ablation studies. Finally, Section 5.6 summarizes my contributions and outlines directions for future work.

5.2 Related Work

World models have shown significant promise in improving sample efficiency through imagination-based training [Ding et al., 2024, Feng et al., 2025, Luo et al., 2024]. Recent efforts proposed to improve sample efficiency in model-free algorithms have been successful through the use of auxiliary objectives [Kim et al., 2023, Liu et al., 2025] or data augmentation [Ma et al., 2023, Liu et al., 2023]. However, even with these improvements, model-free algorithms still require significantly more training samples to achieve results similar to those of world model-based algorithms.

The original world model [Ha and Schmidhuber, 2018] pioneered the idea in the single-agent setting, learning models for imagining future trajectories using autoencoders and recurrent neural networks (RNN) [Rumelhart et al., 1986a]. Following this, several improvements were made in imagined state representations, agent training, and structure and contents of the world model through frameworks such as SimPLE [Kaiser et al., 2020] and the Dreamer series [Hafner et al., 2019, 2020, 2024]. Following the debut of Dreamer, several new world model architectures, IRIS [Micheli et al., 2023], TWM [Robine et al., 2023], TransDreamer [Chen et al., 2022a], and STORM [Zhang et al., 2023], were able to further enhance the Dreamer method with the use of transformers as the core sequence model rather than GRU[Cho et al., 2014] or other RNN-based architectures, as transformers[Vaswani et al., 2017] typically outperform RNNs in a wide variety of sequence modeling tasks due to their ability to model long-range dependencies and enable parallel computation. However, all of these world model architectures only considered single-agent systems and would likely require significant rework to enable success in multi-agent settings.

The existing literature on multi-agent world models is very limited. MAZero [Liu et al., 2024], a multi-agent extension of MuZero [Schrittwieser et al., 2020], is a model-based approach to MARL that closely resembles world models; it uses Monte Carlo Tree Search to plan future trajectories instead of a traditional neural network-based world model, which was computationally expensive and scaled poorly with an increased number of agents or larger action spaces. MBVD and MAMBA were among the first model-based approaches to employ neural network-based world models in MARL. MBVD leverages shared latent rollouts and value decomposition to facilitate coordination, utilizing a GRU-based sequence model [Cho et al., 2014] to generate imagined trajectories. While it introduced an important architectural step forward as one of the earliest world models tailored for multi-agent systems, it still suffers from low sample efficiency, often requiring over one million environment steps to reach performance levels that more recent multi-agent world

model methods can achieve in under 100,000 steps. MAFMBA, a multi-agent extension of DreamerV2, introduces a centralized world model approach that uses joint-agent dynamics to compute a joint-agent state using a GRU-based sequence model, enabling imagination-based learning in partially observable environments. It has demonstrated strong performance across a range of challenging MARL tasks, including those with high agent counts and partial observability, outperforming traditional model-free approaches and earlier model-based methods. While effective for modelling interactions between agents, a downside of the centralized world model can become extremely computationally expensive as the number of agents grows.

CoDreamer [Toledo, 2024] extends DreamerV3 to MARL with an explicit graph neural network-based communication mechanism, enabling agents to share information both within the world model and at the policy level; the paper also reports IDreamer, a strong independent-learning DreamerV3 baseline that treats other agents as part of the environment. Both highlight the value of Dreamer-style imagination in multi-agent settings while differing in whether cross-agent information is modeled explicitly. COMBO [Zhang et al., 2025] is another MARL world model framework that explores egocentric, pixel-based cooperation via a compositional video diffusion world model that factorizes joint behavior and supports planning; it emphasizes long-horizon coordination and uses tree-search-based planning components rather than purely imagination-trained actorcritic updates.

At the time of writing, MARIE [Zhang et al., 2024] introduces the first and only Transformer-based world model for multi-agent systems, which achieved a substantial improvement over MBVD and a marginal improvement MAMBA in a wide range of SMAC maps. It learns independent local dynamics alongside centralized feature aggregation via a Perceiver [Jaegle et al., 2021]. The aggregation module is used to compute contextual information about the entire team from a given agents perspective. These aggregated features are inserted into each agents local token sequence and passed to the shared dynamics model and reward predictor to improve prediction accuracy, allowing for coordination and mitigating nonstationarity during imagination. This method proved successful in enabling multi-agent imagination while being far more memory-efficient than the centralized world model approach found in MAMBA, but had the potential to leak information that may otherwise be unknown due to partial observability. My approach shares the goal of enabling sample efficient multi-agent learning through imagination but takes a complementary route: instead of modeling all agent interactions through centralized aggregation, I explicitly model other agents' behavior with a learned teammate predictor. The computational cost of the teammate predictor remains nearly constant as the number of agents increases, whereas the aggregation module's computational cost does increase with the number of agents. Also, unlike the aggregation module in MARIE, the teammate predictor supervises on other agents actions during world model training using agents' latent, acting as an auxiliary objective for the world model.

MARIE, along with MAMBA, use PPO-style training [Schulman et al., 2017] for their world model and agents; this can be advantageous in multi-agent systems as it ensures that experiences being used to train the world model and agents are recent and do not reflect outdated agent behaviors, but PPO-style training is generally less sample-efficient due to infrequent updates and can suffer from catastrophic forgetting [Kaplanis et al., 2019], particularly when update frequency is increased. Instead, a DreamerV3-style

[Hafner et al., 2024] training framework is adopted, augmented with a recency-biased replay mechanism that emphasizes recent experiences. This setup enables more frequent updates, training both the world model and the agent after each environment step. I hypothesize that this approach enhances sample efficiency due to the higher volume of updates, while mitigating overfitting in agent training by continuously updating imagined trajectories in sync with evolving policies.

5.2.1 Latent Representations

One of the primary distinctions between world model implementations lies in how state information is represented and provided to the agent. Training directly on raw observations can introduce significant noise and dramatically increase computational requirements [Ye et al., 2021, Hafner et al., 2020, Robine et al., 2023]. To address this, most world models now rely on learned latent representations to encode essential features while discarding irrelevant noise and decreasing the dimensionality of the data. Early world model approaches typically employed continuous latent spaces, most commonly learned via standard variational autoencoders (VAEs) [Kingma and Welling, 2013]. As opposed to the Vector Quantized Variational Autoencoder (VQ-VAE), which is a self-supervised method that learns discrete latent representations, achieving comparable performance while producing significantly more compact encodings [van den Oord et al., 2017]; for example, when applied to environments like DeepMind Lab [Beattie et al., 2016], they can represent observations using as few as 27 bits. Furthermore, empirical studies have shown that discrete latent spaces often outperform continuous ones on a wide range of RL tasks, even in model-free algorithms [Meyer et al., 2024].

Nonetheless, DreamerV1 [Hafner et al., 2019] and World Models [Ha and Schmidhuber, 2018] adopted continuous latent spaces for their frameworks. SimPLE [Kaiser et al., 2020] was among the first to instead utilize a VQ-VAE. However, this approach was soon surpassed by DreamerV2 [Hafner et al., 2020], which adopted a Categorical-VAE [Jang et al., 2017] that achieved substantially better performance. Today, several of the most competitive single-agent world models employ Categorical-VAEs as their default. In contrast, Categorical-VAEs are less prevalent in multi-agent world models. MBVD utilizes continuous latent spaces, whereas MARIE employs discrete representations through a VQ-VAE. MAMBA, however, does use a Categorical-VAE. I hypothesize that my use of a Categorical-VAE will provide a slight representational advantage over the existing multi-agent world models that have not yet adopted it.

I present a comparative overview of MARIE, MAMBA, MBVD, and MATWM in Table 5.1, highlighting key distinctions in architectural and training design.

Table 5.1: Comparison of MATWM with leading multi-agent world model architectures

Aspect	MARIE	MAMBA	MBVD	MATWM
Backbone Architecture	Transformer	GRU	GRU	Transformer
Latent Representation	VQ-VAE	Categorical VAE	VAE	Categorical VAE
Critic Type	Centralized	Centralized	Centralized	Semi-centralized
Imagination Style	Hybrid	Centralized	Centralized	Independent
Agent Training Strategy	PPO-style	PPO-style	Deep Q-learning	DreamerV3-style

5.3 Method

This chapter studies multi-agent RL in a finite-horizon partially observable stochastic game (POSG) $\mathcal{G} = (\mathcal{I}, \mathcal{S}, \{\mathcal{A}^{(i)}\}, \{\Omega^{(i)}\}, P, O, \{R^{(i)}\}, \rho_0, \gamma, T)$, where $\mathcal{I} = \{1, \ldots, N\}$ indexes agents. At each time t, the environment is in state $s_t \in \mathcal{S}$. Each agent i receives a private observation $o_t^{(i)} \in \Omega^{(i)}$ sampled from $O(\cdot \mid s_t, i)$ and selects an action $a_t^{(i)} \in \mathcal{A}^{(i)}$. Let $\mathbf{o}_t \triangleq (o_t^{(1)}, \ldots, o_t^{(N)})$ and $\mathbf{a}_t \triangleq (a_t^{(1)}, \ldots, a_t^{(N)})$ denote the joint observation and action. Transitions follow $P(s_{t+1} \mid s_t, \mathbf{a}_t)$. Each agent receives its own reward $r_t^{(i)} = R^{(i)}(s_t, \mathbf{a}_t)$. Episodes start from $s_0 \sim \rho_0$ and terminate after horizon T. I operate in predominantly cooperative, partially competitive settings, where each agent i maximizes $J^{(i)}(\pi) \triangleq \mathbb{E}\left[\sum_{t=0}^{T-1} \gamma^t r_t^{(i)}\right]$.

Due to partial observability, each agent acts based only on its own local information. I make no assumptions about access to other agents observations or actions at decision time. During training I may aggregate trajectories from all agents to fit shared models or critics (centralized training), but each agent executes using only its own history and local computations (decentralized execution). Concretely in my method, a shared world model is trained from the union of agents data, while each policy $\pi^{(i)}$ is executed using per-agent inputs only. Section 5.3 details the specific information flows.

For notation, I use bold symbols for joint variables (e.g., $\mathbf{o}_t$, $\mathbf{a}_t$, $\mathbf{r}_t$) and superscripts for per-agent quantities (e.g., $o_t^{(i)}$, $a_t^{(i)}$, $r_t^{(i)}$, $z_t^{(i)}$). I avoid ambiguous bare a_t or o_t: they are always either joint ($\mathbf{a}_t$, $\mathbf{o}_t$) or per-agent ($a_t^{(i)}$, $o_t^{(i)}$).

All agents in the environment utilize a shared world model, which is trained using an equal distribution of experiences from each agent. In this chapter, the focal agent refers to the agent currently being trained, evaluated, or associated with a given experience, or for whom the world model is generating predictions and trajectories. My approach builds upon STORM [Zhang et al., 2023], a single-agent world model architecture that integrates several state-of-the-art design principles, including transformer-based sequence modeling, symlog reward scaling, KL balance and free bits, and categorical latent representations. MATWM retains STORM's core components but adds key extensions for multi-agent settings. Specifically, MATWM adds a teammate predictor to model other agents behavior, a prioritized replay sampling strategy to maintain up-to-date world model training. These additions enable MATWM to support multi-agent coordination without requiring centralized training or communication. The framework utilized in this chapter is consistent with that of other world model-based algorithms; agents learn a policy via imaginations generated by the world model. This is done via the following three steps that repeat in order:

1. Populate a replay buffer with real experiences using the current policies of the agents.
2. Train the world model using samples from the replay buffer.
3. Sample real experiences from the replay buffer, compute imagined trajectories using the real experiences as a starting point, and update the agents' policies using the imagined trajectories.

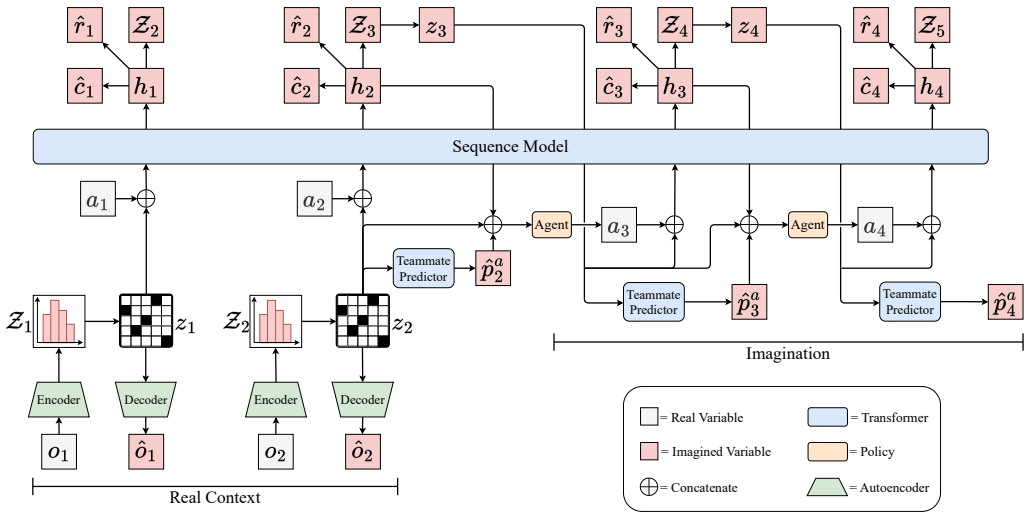

Figure 5.1: Overview of the MATWM architecture. The figure illustrates how real context (past observations and actions) are encoded and used as a starting point for imagination rollouts. Each agent encodes its local observation into a discrete latent state via an autoencoder. These latent states, combined with actions, are processed through an action mixer and a transformer-based sequence model to produce hidden states. The world model then predicts the next latent state, reward, continuation, and optionally, the action mask. The agent then uses the new latent state, predicted teammate action logits and the hidden state to compute a new action. The new action and latent state are then used to repeat this process.

These steps are iterated until the maximum number of allowed steps through the real environment is achieved. Each agent i will have a separate replay buffer, where each entry in the replay buffer contains an observation $o_t^{(i)}$, an action $a_t^{(i)}$, a reward $r_t^{(i)}$, a continuation flag $c_t^{(i)}$ (a Boolean variable denoting whether an episode has terminated), and an action mask $m_t^{(i)}$. Agents' actions will be randomly selected until the replay buffer reaches 1,000 samples, after which agents and the world model will begin training, and agents will use their learned policy to decide actions. After each real environment step, the world model and the agents will undergo one epoch of training. For world model training, I sample batches of length-L sequences from the replay buffer, prioritizing recent samples via an exponentially decaying (geometric) weighting. Specifically, for buffer size K and decay factor $d \in (0, 1)$, each index i is assigned

$$g_i \;=\; d^{K-i}, \qquad w_i \;=\; \frac{g_i}{\sum_{j=1}^{K} g_j}.$$

I then draw start indices without replacement according to w_i, masking out overlapping blocks of length L to ensure non-overlapping sequences in the batch. For agent training, replay buffer entries are sampled uniformly at random with no weighting. After an L-length batch is sampled from each agent's replay memory, the batches from all agents are concatenated and fed to the world model as a single large batch.

5.3.1 World Model

The world model is a collection of predictive models that can replicate the dynamics of the real environment, allowing the agents to interact with and learn from the world model, rather than the real environment directly. My world model architecture comprises several modules that enable the accurate replication of the real environment's dynamics. Figure 5.1 shows a diagram of the MATWM architecture and how a focal agent interacts with it, further explained in Algorithm 3 found in 5.6. I build a world model that is shared by all agents, but only computes the focal agents trajectory using only the focal agent's information.

The world model use an autoencoder to transform each observation $o_t^{(i)}$ into a stochastic latent categorical distribution $\mathcal{Z}_t^{(i)}$. Following prior work [Hafner et al., 2020, Zhang et al., 2023, Hafner et al., 2024], I define $\mathcal{Z}_t^{(i)}$ as consisting of 32 categorical variables, each with 32 discrete classes. The encoder q_ϕ and decoder p_ϕ are implemented as multi-layer perceptrons (MLPs) for vector observations. The latent state $z_t^{(i)}$ is sampled from $\mathcal{Z}_t^{(i)}$ to represent the original observation $o_t^{(i)}$. Since sampling from a categorical distribution is non-differentiable, I use the straight-through gradient estimator [Bengio et al., 2013] to preserve gradients for backpropagation. The encoder and decoder as follows:

$$\begin{aligned} &\text{Encoder:} \quad z_t^{(i)} \sim q_\phi\left(z_t^{(i)} \mid o_t^{(i)}\right) = \mathcal{Z}_t^{(i)}, \\ &\text{Decoder:} \quad \hat{o}_t^{(i)} = p_\phi\left(z_t^{(i)}\right). \end{aligned} \tag{5.1}$$

An action mixer, a single linear transformation denoted by the function $m_\phi(\cdot)$, is utilized to merge the latent state $z_t^{(i)}$ with the agent's action $a_t^{(i)}$, returning an embedded representation $e_t^{(i)}$. This is passed into a vanilla transformer [Vaswani et al., 2017], whose function is denoted by $f_\phi(\cdot)$, acting as the sequence model, which outputs the hidden states $h_{0:T}^{(i)}$ for all $e_{0:T}^{(i)}$. Then, the dynamics predictor, an MLP-based module, denoted by the function $g_\phi(\cdot)$, predicts the next latent state $\hat{z}_{t+1}^{(i)}$ from the $h_t^{(i)}$. Since no real observations are available during imagination, the dynamics predictor is critical for rolling out future states, as I would not be able to use the encoder to compute them. The reward and continuation predictors estimate rewards and continuation flags during imagination. To account for the behaviors of other agents and enable cooperative behavior during both imagination and interaction with the real environment, the teammate predictor utilizes a transformer to infer the action probabilities $\hat{p}^{(i)}$ of all N non-focal agents based on the focal agent's latent state history, where $|\hat{\mathbf{p}}_t^{(j)}| = |\mathcal{A}^{(j)}|$, $\sum_{a \in \mathcal{A}^{(j)}} \hat{\mathbf{p}}_t^{(j)}[a] = 1$. Lastly, the action mask predictor is critical in partially observable or constrained environments like SMAC, where agents are only allowed to execute legal actions. Since imagined rollouts do not have access to real action masks, I train a predictor to estimate them, denoted by $\hat{m}^{(i)}$. This allows us to avoid degenerate scenarios where an agent conducts an illegal action, for which the sequence model may produce an invalid state, thus tainting the entire imagination rollout. I define these world model components as:

$$
\begin{aligned}
&\text{Action mixer:} && e_t^{(i)} = m_\phi^E \left(z_t^{(i)}, a_t^{(i)} \right) \\
&\text{Sequence model:} && h_{0:T}^{(i)} = f_\phi \left(e_{0:T}^{(i)} \right) \\
&\text{Dynamics predictor:} && \hat{\mathcal{Z}}_{t+1}^{(i)} = g_\phi^D \left(\hat{z}_{t+1}^{(i)} \mid h_t^{(i)} \right) \\
&\text{Reward predictor:} && \hat{r}_t^{(i)} = g_\phi^R \left(h_t^{(i)} \right) \\
&\text{Continuation predictor:} && \hat{c}_t^{(i)} = g_\phi^C \left(h_t^{(i)} \right) \\
&\text{Teammate predictor:} && \{\hat{p}_{0:T}^{(i)}\}_{i=1}^N = f_\phi^{\text{act}}(z_{0:T}) \\
&\text{Action Mask predictor:} && \hat{m}_t^{(i)} = g_\phi^M \left(z_t^{(i)} \right)
\end{aligned}
\tag{5.2}
$$

For each agent in the environment, I randomly sample transitions from their replay buffer, then use them to update the world model via the total world model loss function. The world model objective combines seven terms corresponding to observation reconstruction by the autoencoder, reward prediction, continuation prediction, teammate-action prediction, action-mask prediction, latent dynamics prediction, and improving representation learning:

$$
\mathcal{L}(\phi) = \frac{1}{BT} \sum_{n=1}^{B} \sum_{t=1}^{T} \left[\mathcal{L}_t^{\text{rec}}(\phi) + \mathcal{L}_t^{\text{rew}}(\phi) + \mathcal{L}_t^{\text{con}}(\phi) + \mathcal{L}_t^{\text{team}}(\phi) + \beta_1 (\mathcal{L}_t^{\text{mask}}(\phi) + \mathcal{L}_t^{\text{dyn}}(\phi)) + \beta_2 \mathcal{L}_t^{\text{rep}}(\phi) \right], \tag{5.3}
$$

where B is the batch size, T is the batch length, and $\mathcal{L}_t^{\text{rec}}$, $\mathcal{L}_t^{\text{rew}}$, $\mathcal{L}_t^{\text{con}}$, $\mathcal{L}_t^{\text{team}}$, $\mathcal{L}_t^{\text{mask}}$, $\mathcal{L}_t^{\text{dyn}}$, and $\mathcal{L}_t^{\text{rep}}$ are sub-loss functions for different subsets of the world models components.

The reconstruction loss, $\mathcal{L}_t^{\text{rec}}$, is represented by the mean-squared error between the real observation and the predicted reconstruction of the observation, facilitating the training of the encoder and the decoder:

$$
\mathcal{L}_t^{\text{rec}}(\phi) = \left(\hat{o}_t^{(i)} - o_t^{(i)} \right)^2. \tag{5.4}
$$

Next, $\mathcal{L}_t^{\text{rew}}$ and $\mathcal{L}_t^{\text{con}}$ are the components of the world model loss that facilitate the training of the reward and continue predictors, respectively. In $\mathcal{L}_t^{\text{rew}}$, I utilize a symlog two-hot loss, as described in [Hafner et al., 2024]. $\mathcal{L}_t^{\text{con}}$ is the binary cross-entropy loss between the ground truth continuation flag $c_t^{(i)}$ and predicted continuation flag $\hat{c}_t^{(i)}$. These sub-loss functions are computed as follows:

$$
\mathcal{L}_t^{\text{rew}}(\phi) = \mathcal{L}^{\text{sym}} \left(\hat{r}_t^{(i)}, r_t^{(i)} \right), \tag{5.5}
$$

$$
\mathcal{L}_t^{\text{con}}(\phi) = c_t^{(i)} \log \hat{c}_t^{(i)} + \left(1 - c_t^{(i)} \right) \log \left(1 - \hat{c}_t^{(i)} \right). \tag{5.6}
$$

The teammate predictor, given the latent state sequence of the focal agent, outputs a

logits representing the probability of each teammates possible actions. The ground truth teammate actions are used to compute a standard cross-entropy loss. For each teammate $j \in \{1, \cdots, N-1\}$ and each possible action $a \in \{1, \cdots, A\}$, where A is the number of discrete actions, I define:

$$\mathcal{L}_t^{\text{team}}(\phi) = -\sum_{j=1}^{N-1} \sum_{a=1}^{A} \delta(a_{t,j} = a) \log \hat{p}_{t,j}^{(a)}, \tag{5.7}$$

where $a_{t,j}$ is the ground truth action of teammate j at time t, and $\hat{p}_{t,j}^{(a)}$ is the predicted probability of teammate j selecting action a. This objective also serves as an auxiliary signal, as its gradients backpropagate through the encoder, shaping the latent to better capture features predictive of teammates actions.

The action mask loss $\mathcal{L}_t^{\text{mask}}$ is computed via the binary cross-entropy between the predicted action mask $\hat{m}_t^{(i)}$ and the ground-truth binary action mask $m_t^{(i)}$:

$$\mathcal{L}_t^{\text{mask}}(\phi) = m_t^{(i)} \log \hat{m}_t^{(i)} + \left(1 - m_t^{(i)}\right) \log \left(1 - \hat{m}_t^{(i)}\right). \tag{5.8}$$

As with the teammate-prediction loss, this provides a directly supervised auxiliary signal whose gradients backpropagate through the encoder, sharpening the latent representation around action-legality cues and improving imagined rollouts.

The sub-loss functions $\mathcal{L}_t^{\text{dyn}}$ and $\mathcal{L}_t^{\text{rep}}$ are both KullbackLeibler (KL) divergences and differ only in the placement of the stop-gradient operator $\text{sg}(\cdot)$. Here, $\mathcal{L}_t^{\text{dyn}}$ trains the sequence model to predict the next latent accurately, while $\mathcal{L}_t^{\text{rep}}$ encourages the encoder to produce latents that are easier to predict. Following [Hafner et al., 2024, Zhang et al., 2023], I employ free bits [Kingma et al., 2016] to mitigate KL collapse by imposing a small KL floor. Unlike prior work that applies free bits as a single threshold on the aggregate KL, I apply them per latent category, clamping each group-wise KL before summation. Let the discrete latent factorize into G categorical groups, $z_t^{(i)} = [z_{t,1}^{(i)}, \ldots, z_{t,G}^{(i)}]$. I clamp each group-wise KL by a threshold $\lambda_{\text{fb}} = 0.03125$ (in nats) before summing across groups, which prevents well-fit groups from dominating the objective and focuses learning on mismatched components. The losses are:

$$\mathcal{L}_t^{\text{dyn}}(\phi) = \sum_{g=1}^{G} \max\Big(\lambda_{\text{fb}},\, \text{KL}\big[\,\text{sg}\big(q_\phi\Big(z_{t+1,g}^{(i)} \mid o_{t+1}^{(i)}\Big)\big) \;\big\|\; g_\phi^D\Big(z_{t+1,g}^{(i)} \mid h_t^{(i)}\Big)\,\big]\Big), \tag{5.9a}$$

$$\mathcal{L}_t^{\text{rep}}(\phi) = \sum_{g=1}^{G} \max\Big(\lambda_{\text{fb}},\, \text{KL}\big[q_\phi\Big(z_{t+1,g}^{(i)} \mid o_{t+1}^{(i)}\Big) \;\big\|\; \text{sg}\big(g_\phi^D\Big(z_{t+1,g}^{(i)} \mid h_t^{(i)}\Big)\big)\big]\Big). \tag{5.9b}$$

5.3.2 Training Structure

To train agents using imagined experience, I start from a context window of real observations and actions. The world model uses this context to produce an initial latent state $z_0^{(i)}$

and hidden state $h_0^{(i)}$ for the focal agent i. During each imagination step, the focal agent samples an action $a_t^{(i)}$ based on its current state and the predicted action distributions of its teammates $\{\hat{\mathbf{p}}_t^{(j)}\}_{j\neq i}$, produced by the teammate predictor. When enabled, the action mask predictor $\hat{m}_t^{(i)}$ is used to enforce action feasibility constraints. The action $a_t^{(i)}$ is then passed to the world model to predict the next latent state $\hat{z}_{t+1}^{(i)}$, hidden state $h_{t+1}^{(i)}$, reward $\hat{r}_t^{(i)}$, and termination/continuation signal $\hat{c}_t^{(i)}$, which are stored in temporary rollout buffers. The resulting trajectories, stored in that temporary rollout buffer, allow the agent policies to learn entirely from imagination, without requiring new environment interactions.

Each agent is trained independently via a shared world model. Agents do not interact with one another directly; instead, they interact with imagined versions of other agents whose behavior is captured within the state information and by the teammate predictor. The rationale behind this approach is that the world model will treat non-focal agents as it would any other non-deterministic entity, similar to the non-agent adversaries in a stochastic single-agent game, predicting their behavior and thus impact on the following state(s) to the best of its ability. The alternative, to have agents coexist in and interact with the same imagined environment, poses several challenges. Firstly, the world model may produce mutually inconsistent or illegal states for two or more agents, causing agents to learn from experiences that could never possibly occur in the real environment. Secondly, computing a joint trajectory would cause the memory requirements of the world model to scale poorly with the number of agents, making it impractical for environments involving many agents. Nonetheless, I hypothesize that this training structure could exacerbate the nonstationarity issue, as the world model might be displaying outdated behaviors of the agents during imagination. Again, I alleviate this issue by using a replay memory structure that prioritizes sampling recent memories when training the world model. I share the full training algorithm in 5.6.

The state $s_t^{(i)}$ for each agent is formed by concatenating the predicted action distributions $\{\hat{\mathbf{p}}_t^{(j)}\}_{j\neq i}$ of all non-focal agents and its own $z_t^{(i)}$ and $h_t^{(i)}$. Agents are trained through an actor-critic learning algorithm. Each agent has its own critic, which only takes in the local state $s_t^{(i)}$ of that agent along with that agent's action $a_t^{(i)}$. A centralized critic would not be feasible here because during imagination, the agents' states are very likely to become desynchronized, so the actions conducted by an agent in its imagined state may not be transferable to another agent's imagined state, even in the same time step; additionally, they may have imagined states that are conflicting and cannot legally exist at the same time. This is why I use the teammate predictor to predict what the non-focal agents would do in the focal agent's current state, allowing us to have a semi-centralized critic that does not require having direct access to non-focal agent information. The Critic and Actor are denoted by:

$$\begin{aligned} &\text{Critic}: \quad V_{\psi^{(i)}}\left(s_t^{(i)}\right) \approx \mathbb{E}_{\pi_{\theta^{(i)}}, p_\phi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}^{(i)}\right], \\ &\text{Actor}: \quad a_t^{(i)} \sim \pi_{\theta^{(i)}}\left(a_t^{(i)} \mid s_t^{(i)}\right). \end{aligned} \tag{5.10}$$

$$\mathcal{L}\big(\theta^{(i)}\big) = \frac{1}{BL}\sum_{n=1}^{B}\sum_{t=1}^{L}\Big[-\operatorname{sg}\big(\frac{\tilde{A}_t^{(i)}}{S^{(i)}}\big)\ \ln \pi_{\theta^{(i)}}\Big(a_t^{(i)} \mid s_t^{(i)}\Big) - \eta\, H\Big(\pi_{\theta^{(i)}}\Big(a_t^{(i)} \mid s_t^{(i)}\Big)\Big)\Big],$$
$$\mathcal{L}\big(\psi^{(i)}\big) = \frac{1}{BL}\sum_{n=1}^{B}\sum_{t=1}^{L}\Big[\big(V_{\psi^{(i)}}\Big(s_t^{(i)}\Big) - \operatorname{sg}\Big(G_t^{\lambda,(i)}\Big)\big)^2 + \big(V_{\psi^{(i)}}\Big(s_t^{(i)}\Big) - \operatorname{sg}\Big(V_{\psi^{(i)},\mathrm{EMA}}\Big(s_t^{(i)}\Big)\Big)\big)^2\Big]. \tag{5.11}$$

Where $H(\cdot)$ represents the entropy of the policy distribution, and the constants η and L denote the entropy regularization coefficient and the imagination horizon, respectively. The λ-return $G_t^{\lambda,(i)}$ is recursively defined [Sutton and Barto, 1998] over the imagination horizon as follows:

$$\begin{aligned} G_t^{\lambda,(i)} &\doteq r_t^{(i)} + \gamma\, c_t^{(i)}\Big[(1-\lambda)\, V_{\psi^{(i)}}\Big(s_{t+1}^{(i)}\Big) + \lambda\, G_{t+1}^{\lambda,(i)}\Big], \\ G_L^{\lambda,(i)} &\doteq V_{\psi^{(i)}}\Big(s_L^{(i)}\Big). \end{aligned} \tag{5.12}$$

In this work, the normalization factor $S^{(i)}$ used in the policy loss $\mathcal{L}(\theta^{(i)})$ is computed as the difference between the 95th and 5th percentiles of the agent-wise advantage $\hat{A}^{(i)}$ within the batch. This differs from prior implementations [Hafner et al., 2024, Zhang et al., 2023], which compute S from the 95th and 5th percentiles of the λ-return. I made this change because the policy update is proportional to the policy gradient term $-\log \pi \cdot A$. By normalizing the advantage directly, I keep step sizes aligned with the actual learning signal, so update sizes remain consistent as the critic improves and raw advantages shrink. In contrast, a λ-return normalizer mixes in effects from varying episode lengths and reward magnitudes, factors orthogonal to policy improvement, often making updates too conservative late in training or too aggressive early on. I define the normalization factor $S^{(i)}$ as:

$$S^{(i)} \;=\; \operatorname{percentile}\Big(\hat{A}^{(i)}, 95\Big) \;-\; \operatorname{percentile}\Big(\hat{A}^{(i)}, 5\Big), \tag{5.13}$$

where

$$\hat{A}_t^{(i)} \;=\; G_t^{\lambda,(i)} \;-\; V_{\psi^{(i)}}\Big(s_t^{(i)}\Big). \tag{5.14}$$

I then compute centered advantages:

$$\tilde{A}_t^{(i)} \;=\; \hat{A}_t^{(i)} \;-\; \mu^{(i)}, \qquad \mu^{(i)} \;=\; \frac{1}{|\mathcal{B}^{(i)}|}\sum_{t\in\mathcal{B}^{(i)}} \hat{A}_t^{(i)}. \tag{5.15}$$

Where $\mathcal{B}^{(i)}$ is the set of valid (non-terminal) timesteps for agent i in the minibatch.

To regularize the value function, I maintain an exponential moving average (EMA) of the critic parameters $\psi^{(i)}$. As defined in Equation (16), $\psi_t^{(i)}$ represents the current critic parameters, σ is the decay rate, and $\psi_{t+1}^{(i),\mathrm{EMA}}$ denotes the updated EMA. This technique

helps stabilize training and mitigate overfitting:

$$\psi_{t+1}^{(i),\mathrm{EMA}} = \sigma\, \psi_t^{(i),\mathrm{EMA}} + (1-\sigma)\, \psi_t^{(i)}. \tag{5.16}$$

5.4 Experiments

MATWM is evaluated on the StarCraft Multi-Agent Challenge (SMAC) [Samvelyan et al., 2019], a vector-observation benchmark of partially observable micromanagement scenarios requiring coordinated control to defeat built-in opponents. I use 8 maps (7 *easy*, 1 *hard*). Each agent observes features of allied and enemy units within sight range, including distance, relative position (x,y), health, shield, and unit type. The discrete action space consists of five movement actions, `attack[enemy_id]`, `stop`, and `no-op`. The default shaped reward is used, which reflects damage dealt, enemy units killed, and a bonus for winning the battle. Agents are evaluated on their ability to win the battle rather than cumulative reward.

Following prior work [Samvelyan et al., 2019, Zhang et al., 2024], I report the median and standard deviation over four random seeds; evaluation is mean win rate across 50 episodes. *Easy* maps are trained for 50K steps and *hard* maps for 200K. While 100K is standard for *easy* maps, many existing methods reach 100% win rate under that budget [Zhang et al., 2024, Egorov and Shpilman, 2022]; I use 50K to mitigate multi-way ties. MATWM hyperparameter settings for these experiments are provided in 5.6.

MATWM is compared against several state-of-the-art baselines in MARL. First, I evaluate it against three world model architectures: MARIE, MAMBA, and MBVD, all of which represent the current frontier in world models for multi-agent systems. In addition to these model-based baselines, I also compare MATWM with a set of strong model-free algorithms, providing a broad and representative benchmark across different learning paradigms. QMIX [Rashid et al., 2020] is a popular value-decomposition method that factors the joint action-value function into individual agent utilities under a monotonicity constraint. QPLEX [Wang et al., 2021] extends QMIX by incorporating a duplex dueling architecture that enables more expressive value functions. MAPPO [Yu et al., 2022] is a centralized variant of Proximal Policy Optimization [Schulman et al., 2017] adapted for multi-agent settings, using a shared critic across agents.

5.4.1 SMAC Results

Table 5.2: Win rate as a % comparison across various SMAC maps: Median (Std).

Map	Steps	MATWM		MARIE		MAMBA		MBVD		MAPPO		QMIX		QPLEX	
		Med	Std	Med	Std	Med	Std	Med	Std	Med	Std	Med	Std	Med	Std
2m_vs_1z	50K	**100.0**	0.4	95.0	4.4	91.0	6.2	41.0	20.7	51.0	10.3	62.0	8.9	70.0	13.0
2s_vs_1sc	50K	**99.5**	0.8	90.0	9.1	80.0	7.3	0.0	1.2	18.0	7.6	8.0	4.4	13.0	6.8
2s3z	50K	**79.0**	6.2	71.0	8.6	66.0	12.1	28.0	17.5	13.0	3.0	17.0	4.3	36.0	5.8
3m	50K	**83.0**	7.4	78.0	8.1	68.0	7.7	60.0	9.2	54.0	6.3	61.0	10.2	58.0	4.9
8m	50K	71.0	5.9	**72.0**	7.1	68.0	6.4	52.0	18.9	38.0	4.9	60.0	18.2	59.0	9.4
MMM	50K	**9.0**	4.7	1.0	1.6	3.0	3.5	0.0	0.0	0.0	0.0	0.0	0.0	3.0	2.7
so_many_baneling	50K	**85.0**	7.9	73.0	11.4	66.0	14.2	27.0	12.3	31.0	7.6	40.0	6.1	52.0	8.8
2c_vs_64zg	200K	**21.0**	5.2	14.0	8.2	0.0	0.8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Mean		**68.4**		61.8		55.5		26.0		25.6		31.0		36.4	

I evaluate the framework on the StarCraft Multi-Agent Challenge, a widely adopted benchmark for multi-agent coordination under partial observability. The model is compared against both model-free and model-based baselines across a diverse set of 8 scenarios with varying agent counts, including seven *easy* maps and one *hard* map. The results of these experiments are displayed in Table 5.2. I observe that, given a very small sample budget, MATWM outperforms all model-free baselines and even MBVD by a substantial margin, while also outperforming MAMBA and MARIE by lesser, but still decisive margins. On several *easy* maps, such as 2m_vs_1z, 2s_vs_1sc, and so_many_baneling, the model exhibits near-optimal behavior, achieving win rates of 85% or higher. In 2s3z, where tighter coordination is required, MATWM remains ahead, while model-free baselines only barely learn a winning strategy in the 50K sample budget. In MMM, the hardest of the *easy* maps, it is difficult for even the world model baselines to consistently learn a winning strategy within the sample budget; MATWM, however, is able to do so more consistently than other world model baselines. Particularly in the *hard* map, 2c_vs_64zg, model-free methods exhibit no success, unable to learn a winning strategy, while MATWM again continues to be able to do so consistently. Compared to other world model baselines, I found that MATWM may often require more time to learn a winning strategy on some of the *easy* maps. However, once it does, it rapidly refines and exploits that strategy, achieving significantly higher win rates in a shorter span. Also, I found that early training can often have periodic bouts of high then low winrates, likely induced by nonstationarity. But it is consistently able to recover, likely due to the recency-biased world model training method. These results establish MATWM as a leading model in low-data, multi-agent settings, especially where coordination and partial observability are critical.

5.4.2 Ablation Study

To evaluate the contribution of MATWMs key mechanisms, I perform ablations on three components: the teammate predictor, prioritized experience replay (PER), and the action mask predictor. For each, I disable the mechanism and measure performance across representative SMAC tasks. Results are summarized in Table 5.3.

Teammate Predictor. Removing the teammate predictor causes a large perfor-

mance drop, particularly in scenarios requiring strong coordination, like 8m and so_many_baneling. In these settings, win rates often fall harder than in 3m, a lower agent environment, confirming that explicitly modeling teammate behavior is crucial for cooperative play. In a smaller or loosely coupled tasks, 3m, performance remains competitive but still lags behind the full model. Since the teammate predictor receives the same features as the sequence model, it may not always discover new information that is not already captured by the sequence model, the main benefit is more likely the introduction of an auxiliary loss that shapes the latent representation through backpropagation. Also, upon further inspection, I observed that reward predictor accuracy benefits most from the inclusion of the teammate predictor, which is likely a result of the improved latent representation.

Prioritized Replay. Replacing PER with uniform sampling also hurts performance, especially in harder tasks like 8m. Without prioritization, the replay buffer contains more outdated transitions that reflect obsolete teammate behaviors, which amplifies nonstationarity. PER mitigates this by biasing toward recent trajectories, keeping the world model better aligned with evolving policies. But, in simpler settings like 3m, where behavior stabilizes quickly, the effect is less pronounced. This removal also has consequences on the effectiveness of the teammate predictor and thus the latent, which in turn will affect the entire system's functionality. Early in training when actions are noisy and random, the teammate predictor is not only less accurate but also much of this random noise can backpropogate into the latent representation and potentially slightly corrupt the encoder and sequence model. Without PER, the world model is training on these noisy experiences more frequently.

Action Mask Predictor. Disabling the action mask predictor results in degraded stability and performance. During imagination, the absence of mask prediction allows the agent to take illegal actions, which can corrupt imagined rollouts and propagate errors through the trajectory. During training, this resulted in very volatile winrate swings, which can be shown in the standard deviation of the results. By enforcing feasibility constraints, the mask predictor reduces rollout errors and stabilizes training, particularly in environments where action legality is critical. When invalid actions are masked, their logits are set to a large negative value, making their softmax probabilities near zero. This artificially lowers the policy entropy, which in turn encourages the actor to increase entropy because the negative entropy term is part of the actor loss. This is an intended mechanism that helps prevent entropy collapse. Without the action mask predictor, however, the actor may become overly deterministic earlier in training and forgo sufficient exploration. Design choices and hyperparameter settings were tuned for the full MATWM architecture, so it is likely that this ablated variant could achieve stronger results with dedicated retuning and better design decisions regarding exploration.

Overall, these results show that MATWMs performance stems from the interplay of multiple design elements: the teammate predictor for coordination, PER for handling nonstationarity, and the action mask predictor for stabilizing imagined rollouts under legality constraints.

Table 5.3: Ablation Study: Impact of Component Removal on Performance (SMAC win rate %, mean ± std over 4 seeds).

Environment	Full MATWM	– Teammate Predictor	– PER	– Action Mask Predictor
3m	**83.0 ± 7.4**	71.0 ± 5.5	74.0 ± 2.8	51.0 ± 8.2
8m	**71.0 ± 5.9**	51.0 ± 3.7	52.0 ± 2.5	29.0 ± 6.1
so_many_baneling	**85.0 ± 7.9**	68.0 ± 4.8	59.0 ± 3.3	44.0 ± 8.6

5.5 Discussion

The experiments highlight the importance of design choices such as recency-biased replay, explicit teammate modeling, and action-mask prediction in stabilizing imagination-based rollouts. The results suggest that independent imagination frameworks represent a promising direction for low-sample multi-agent reinforcement learning. Although they introduce challenges, most notably exacerbating non-stationarity and providing less implicit modeling of agent interactions, these results with MATWM demonstrate that these issues can be effectively mitigated through careful architectural and training design. In contrast, the scalability and computational demands of centralized joint imagination make it a far less practical solution for large-scale multi-agent systems. With that being said, throughout this study I found several behaviors of independent imagination that may help future work. First, when rewards are shared in partially observable environments, credit assignment becomes much harder. Any errors from the reward predictor compound with the already noisy reward signals that are generated outside the agents local observation window. I found that shortening γ and λ can help make training more stable in some environments with higher agent counts. Second, reward sparsity becomes extremely difficult to work with. In large imagination batches, the rare trajectories that actually lead to an important reward can get washed out twice: first by the reward predictor, which learns an averaged signal over mostly low-reward rollouts, and then again by the critic, which averages those predictions across the batch. As a result, the important strategy that produced the sparse reward is treated as statistically insignificant and never gets reinforced. Thus, I found that there needs to be a careful balance in how large the imagination batches are, particularly in SMAC environments. If the batch is too small, training becomes noisy and high-variance. But if the batch is too large, training becomes overly smooth, and the useful high-value trajectories are averaged away before the agent can learn from them. In my experience, very large batch sizes prevented learning optimal strategy in some environments. Lastly, while independent imagination gives us a much more computationally efficient training pipeline, there is very likely an upper limit on how many agents can share a single world model. I found that environments with large agent-action spaces are highly sensitive to hyperparameter choices, with small deviations often leading to training instability or complete failure. Past some team or action size, I expect one of two failures: either the model becomes too fragmented and no longer has enough capacity to meaningfully capture each agents dynamics, or it starts averaging those dynamics together even in the presence of an agent disambiguation mechanism. In both cases, the shared model stops behaving like a consistent predictor for any one agent, which directly hurts the quality of imagined rollouts.

5.6 Conclusion

This chapter introduced MATWM, a transformer-based multi-agent world model that extends the STORM architecture from the single-agent domain to multi-agent reinforcement learning. MATWM leverages imagination-based training, a decentralized rollout scheme, and a novel teammate predictor to capture coordination in partially observable settings. It incorporates key design elements from state-of-the-art world modelssuch as symlog two-hot rewards, KL balance with free bits, and percentile return normalization-while adapting them to the multi-agent setting.

Across a diverse set of SMAC environments, MATWM achieves state-of-the-art performance with remarkable sample efficiency, reaching near-optimal performance in as few as 50K environment interactions, one of the lowest training budgets reported in the MARL literature. The experiments highlight the importance of design choices such as prioritized replay, explicit teammate modeling, and action-mask prediction in stabilizing imagination-based rollouts. I believe MATWM provides a strong foundation for advancing multi-agent world modeling.

There are several avenues for future work. While single-agent world models often transfer across tasks with little or no retuning, this is less feasible in MARL: varying agent counts, non-stationarity, and credit assignment create substantial environmental heterogeneity and greater hyperparameter sensitivity. MARL domains often resist a one-size-fits-all solution, this is especially true for independent imagination frameworks. Going forward, I plan to tailor and optimize the architecture for more specific MARL subproblems, including extremely sparse-reward regimes, severe partial observability, and scenarios with substantially larger agent populations.

Appendix A: Imagination Algorithm

Algorithm 3: Imagined Rollouts with Teammate Prediction

Input: Agents $\mathcal{I} = \{1, \dots, N\}$, world model WM, context $\{(o_t^{(i)}, a_t^{(i)})\}_{t=0}^{L-1}$, context length L, batch size B, imagination horizon H

Output: Buffers of $\{z_\tau^{(i)}, h_\tau^{(i)}, a_\tau^{(i)}, \hat{r}_\tau^{(i)}, \hat{c}_\tau^{(i)}, \hat{p}_\tau^{(-i)}, \hat{m}_\tau^{(i)}\}$

Initialize per-agent buffers for latents z, hiddens h, actions a, rewards $\hat{r}$, continuations $\hat{c}$, teammate logits $\hat{p}^{(-i)}$, and (optional) masks $\hat{m}$

`// Encode context and bootstrap the recurrent state`

foreach $i \in \mathcal{I}$ **do**

  Encode $o_t^{(i)} \mapsto z_t^{(i)}$ for $t = 0{:}L-1$ with WM's encoder

for $t = 0$ **to** $L-1$ **do**

  foreach $i \in \mathcal{I}$ **do**

    $(\hat{z}_{t+1}^{(i)}, h_{t+1}^{(i)}, \hat{r}_t^{(i)}, \hat{c}_t^{(i)}) \leftarrow WM.\textsc{Predict}\big(z_t^{(i)}, a_t^{(i)}\big)$

Store $\{z_L^{(i)}, h_L^{(i)}\}_i$ as the rollout start

`// Imagined rollout for` H `steps`

for $t = 0$ **to** $H-1$ **do**

  foreach $i \in \mathcal{I}$ **do**

    `// Teammate prediction (from focal agent` i`'s latent)`

    $\hat{p}_t^{(-i)} \leftarrow WM.\textsc{TeammatePredictor}\big(z_{L+t}^{(i)}\big)$

    if *masking enabled* **then**

      $\hat{m}_t^{(i)} \leftarrow WM.\textsc{MaskPredictor}\big(z_{L+t}^{(i)}\big)$

      Sample $a_{L+t}^{(i)} \sim \pi^{(i)}\big(z_{L+t}^{(i)}, h_{L+t}^{(i)}, \hat{p}_t^{(-i)}, \hat{m}_t^{(i)}\big)$

    else

      Sample $a_{L+t}^{(i)} \sim \pi^{(i)}\big(z_{L+t}^{(i)}, h_{L+t}^{(i)}, \hat{p}_t^{(-i)}\big)$

    Buffer $\{a_{L+t}^{(i)}, \hat{p}_t^{(-i)}\}$ (and $\hat{m}_t^{(i)}$ if used)

  foreach $i \in \mathcal{I}$ **do**

    $(\hat{z}_{L+t+1}^{(i)}, h_{L+t+1}^{(i)}, \hat{r}_{L+t}^{(i)}, \hat{c}_{L+t}^{(i)}) \leftarrow WM.\textsc{Predict}\big(z_{L+t}^{(i)}, a_{L+t}^{(i)}\big)$

    Set $z_{L+t+1}^{(i)} \leftarrow \hat{z}_{L+t+1}^{(i)}$ and buffer $\{\hat{r}_{L+t}^{(i)}, \hat{c}_{L+t}^{(i)}, z_{L+t+1}^{(i)}, h_{L+t+1}^{(i)}\}$

return All per-agent buffers $\{z, h, a, \hat{r}, \hat{c}, \hat{p}^{(-i)}, \hat{m}\}$

Appendix B: Training Algorithm

Algorithm 4: Joint Training of World Model and Agents in MATWM

Input: Environment $\mathcal{E}$, world model WM, actorcritic agents $\{\pi^{(i)}\}_{i=1}^{N}$, replay buffers $\{RB^{(i)}\}_{i=1}^{N}$

Parameters *Total steps T, training intervals t_{WM}, t_{π}, batch sizes, context lengths*

Initialize environment $\mathcal{E}$ and per-agent buffers, context queues

for $t = 1$ **to** T **do**

  foreach *agent* $i \in \{1, \ldots, N\}$ **do**

    if $RB^{(i)}$ *is ready* **then**

      Encode recent observations $o_t^{(i)}$ into latent $z_t^{(i)}$ using WM

      Predict teammate behavior $\hat{a}_t^{(-i)}$ using WM's teammate predictor

      Sample action $a_t^{(i)} \sim \pi^{(i)}\left(z_t^{(i)}, \hat{a}_t^{(-i)}\right)$

    else

      Sample $a_t^{(i)}$ randomly (optionally with action mask $m_t^{(i)}$)

  Step environment $\mathcal{E}$ with $\{a_t^{(i)}\}_{i=1}^{N}$, collect $\{o_{t+1}^{(i)}, r_t^{(i)}, c_t^{(i)}\}$

  Store transitions $\left(o_t^{(i)}, a_t^{(i)}, r_t^{(i)}, c_t^{(i)}, m_t^{(i)}, a_t^{(-i)}\right)$ in each $RB^{(i)}$

  if *all episodes terminated at t* **then**

    Reset environment and context buffers

  if $t \bmod t_{WM} = 0$ *and all $RB^{(i)}$ ready* **then**

    Sample batches $\{o_{t:t+L}^{(i)}, a_{t:t+L}^{(i)}, r_{t:t+L}^{(i)}, c_{t:t+L}^{(i)}, a_{t:t+L}^{(-i)}\}$ from each $RB^{(i)}$

    Train WM on the combined batch

  if $t \bmod t_{\pi} = 0$ *and all $RB^{(i)}$ ready* **then**

    Use WM to imagine trajectories $\{\hat{o}_{\tau}^{(i)}, \hat{r}_{\tau}^{(i)}, \hat{c}_{\tau}^{(i)}, \hat{a}_{\tau}^{(-i)}\}$ for each agent i

    Train each policy $\pi^{(i)}$ with imagined rollouts

Appendix C: Reproducibility

Code for MATWM is available on GitHub [2]. For all SMAC experiments, I use Starcraft II version SC2.4.1.2.60604. Please note that using different versions may yield slightly different results. Hyperparameter settings for MATWM are shown in Table 5.4. The settings for all baselines are taken from [Zhang et al., 2024] for the SMAC experiments and [Papadopoulos et al., 2025] for the PettingZoo and MeltingPot experiments.

[2] github.com/azaddeihim/matwm

[3] These settings are for environments with three or less agents. For environments with four to six agents and for 2c_vs_64z, I use a batch size of 768 with an imagination horizon of 12. For environments with seven or more agents, I use a batch size of 1024 with an imagination horizon of 8.

Table 5.4: MATWM Hyperparameter Settings

Hyperparameter	**Value**
Max sequence length	64
Hidden dimension	64
Number of layers	2
Number of attention heads	8
Latent dimension size	32
Number of categories per latent	32
World model train batch length	64
World model train batch size	16
World Model Learning rate	3×10^{-5}
Actor+Critic Learning rate	3×10^{-4}
Optimizer	Adam
Gradient clipping agent	20.0
Gradient clipping world model	20.0
Replay buffer size	50,000
Replay sampling priority decay	0.998
KL loss weight (β_1)	0.5
Representation loss weight (β_2)	0.1
Imagination horizon	$16^{\ 3}$
Agent train batch size	512^3
Imagination context length	8
Train world model every n steps	1
Train agent every n steps	1
MLP Encoder Hidden dim	64
MLP Encoder Hidden layers	3
CNN Encoder Kernel Size	4
CNN Encoder Stride	2
CNN Encoder Layers	4
γ	0.985
λ	0.95

Appendix D: PettingZoo & MeltingPot

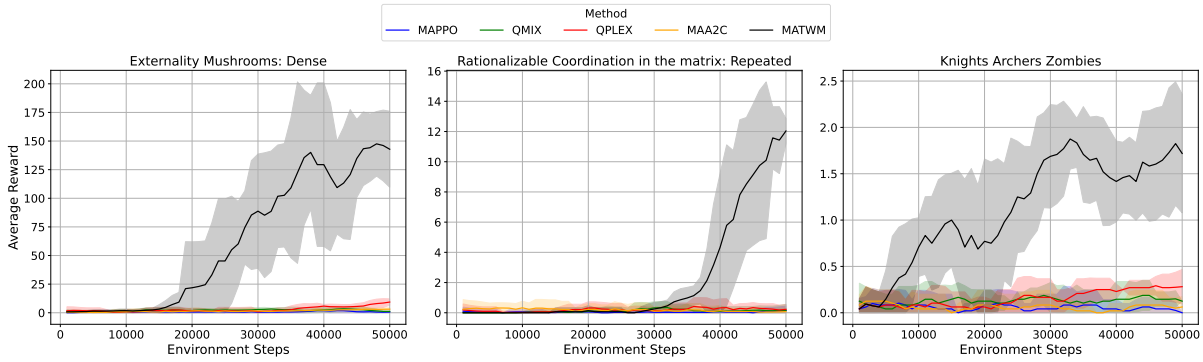

Figure 5.2: Curves of average reward obtained across all agents for each method on three representative PettingZoo and MeltingPot environments. Curves are smoothed for visual clarity of the graphs. The x-axis represents the number of steps taken in the environment, while the y-axis shows the corresponding average reward across all agents.

Table 5.5: Performance comparison across various PettingZoo Butterfly environments: Median (Std).

Environment	Steps	MATWM		MAA2C		MAPPO		QMIX		QPLEX	
		Med	Std	Med	Std	Med	Std	Med	Std	Med	Std
Cooperative Pong	50K	**54.5**	25.1	9.5	3.1	5.5	1.8	10.2	2.4	11.7	2.9
Knights Archers Zombies	50K	**1.75**	1.3	0.0	0.1	0.0	0.2	0.12	0.3	0.25	0.3
Pistonball	50K	**92.6**	2.3	2.5	0.0	-1.3	0.0	4.1	0.0	8.4	0.0

Table 5.6: Performance comparison across various MeltingPot Environments: Median (Std).

Environment	Steps	MATWM		MAA2C		MAPPO		QMIX		QPLEX	
		Med	Std	Med	Std	Med	Std	Med	Std	Med	Std
Chicken in the matrix: Arena	50K	**21.5**	2.4	1.3	0.8	2.1	1.1	3.1	1.4	2.7	0.9
Coop Mining	50K	**19.0**	4.1	9.5	6.1	1.4	0.7	1.0	0.6	1.7	0.5
Externality Mushrooms: Dense	50K	**146.8**	18.5	1.9	3.1	1.0	3.2	2.2	6.7	9.2	9.1
Gift Refinements	50K	**75.0**	25.1	9.5	6.1	6.5	10.8	14.2	16.4	23.7	21.9
Pure Coordination in the matrix: Repeated	50K	**6.8**	0.7	0.0	0.4	0.0	0.0	0.0	0.0	0.0	0.1
Rationalizable Coordination in the matrix: Repeated	50K	**12.2**	0.5	0.3	0.9	0.0	0.5	0.2	0.3	0.4	0.8
Stag Hunt in the matrix: Arena	50K	**7.2**	2.5	1.0	0.7	0.7	0.4	0.0	0.7	1.2	1.0

I evaluate the performance of MATWM on image-based multi-agent environments using the PettingZoo and MeltingPot benchmark suites. World model baselines are not included for PettingZoo and MeltingPot due to the absence of published adaptations or results for use in image-based environments. These experiments were done using an early version of MATWM, which did not include an action mask predictor, and did not utilize the updated actor critic loss functions, nor the updated free-bits formulation. The results in Tables 5.5 and 5.6 highlight the strong generalization and sample efficiency of the model in visually complex settings. In the PettingZoo scenarios, MATWM consistently outperforms all four model-free baselines across all evaluated tasks. For instance, despite Pistonball's staggering 20 agents, MATWM is still able to achieve near-optimal performance with a very small sample budget, outperforming model-free baselines by a significant margin. I observe a similar trend in Cooperative Pong and Knights Archers Zombies, a substantial margin over baselines, whereas the model-free baselines perform roughly similar to a random policy. This showcases the models capacity to learn effective cooperative behaviors from visual observations under tight sample constraints. MeltingPot results further reinforce these trends. The model consistently achieves high rewards across all scenarios, regardless of the number of agents, which range between two and eight, compared to scores from all baselines. Even in more subtle or sparse-reward environments like Pure Coordination in the matrix: Repeated, MATWM is still able to quickly learn strategies that achieve high rewards. Similarly to the PettingZoo experiments, model-free baselines behave at or only slightly better than a random policy on nearly all evaluated environments, highlighting their inability to learn meaningful behavior under limited training budgets. Previous studies conducted on PettingZoo Butterfly environments suggest that it would take nearly one million or more environment steps for any of the model-free baselines to achieve comparable results to what MATWM did in just 50k [Papadopoulos et al., 2025, Formanek et al., 2023]. Figure 5.2 shows performance curves for MATWM and baseline methods evaluated at after every 1000 environment steps across three representative environments: one from PettingZoo and two from MeltingPot. These results demonstrate the benefits of incorporating imagined

trajectories for sample efficiency as baselines often fail to make progress. One recurring pattern I observed during training, also visible in the figure, is that MATWM's performance occasionally dips before stabilizing again. Although this may be environment dependent, I hypothesize that such fluctuations may actually stem from the world model temporarily falling behind the rapidly updating agent policies. This mismatch can momentarily destabilize training and reflects a known limitation of the independent world model approach, even when using prioritized replay buffers.

Chapter 6

Conclusion and Future Work

This thesis set out to explore and advance two underdeveloped but critically important areas of deep learning: multivariate time series (MTS) modeling and multi-agent reinforcement learning (MARL). Both areas are highly relevant to real-world applications such as electrical power systems, yet they lag behind their more established single-variable and single-agent counterparts in terms of methodological maturity, scalability, and robustness. Beginning from a concrete application in the electrical power systems domain, the AC optimal power flow (ACOPF) problem, the research gradually shifted toward the development of novel deep learning architectures that can better capture the complex, structured relationships inherent in other power systems tasks and beyond. This work has the potential to benefit researchers across several different areas of deep learning and applied machine learning.

Three main contributions were presented, each addressing a distinct problem:

- **Graph Neural Networks for ACOPF Warm-Starts:** I developed a GNN-based reinforcement learning framework that learns to generate high-quality warm-starts for classical ACOPF solvers, significantly reducing solution time without sacrificing optimality.
- **Spatio-Temporal Transformer for MTS Forecasting:** I introduced a Transformer-based architecture that employs relative spatio-temporal embeddings to jointly model temporal dependencies and variable interactions, yielding state-of-the-art forecasting accuracy on multiple benchmark datasets.
- **Transformer World Model for MARL:** I proposed a world model architecture for MARL that enables agents to "imagine" and plan in latent space, dramatically improving sample efficiency and coordination under partial observability.

While promising, the approaches presented still have issues that should be addressed. Among all research areas addressed in this thesis, there was a common problem: deep learning approaches do not scale-well with input size. In my ACOPF solution, scaling to larger power systems is mostly tolerable, as the GNN's memory requirements does not scale poorly with increasing graph sizes. Nonetheless, computational time does scale poorly with increasing graph sizes, and while that likely would not play a large factor during inference, as it should still be able to operate in real-time, even with systems with 1000s of buses, training time could increase significantly. Additionally, as graphs become larger and more complex, it may be necessary to also increase the parameter count of the GNN to compensate for the increased complexity of the nodes' relationships. In the STTRE chapter, the proposed approach was memory efficient but scaled poorly with input size due to the way that some of the input embeddings were designed. This would make it quite difficult for this approach to handle MTS data which has many variables and many timesteps. I believe this would be solvable via minor augmentations in the input embeddings, and should only incur a negligible decrease in accuracy since I did confirm in some ablations that input embeddings only offered a minor improvement anyway.

Another common theme that is observed in this thesis is that attention is a powerful and versatile mechanism. This is not by any means an unpopular opinion, but this research does well in upholding the notion. In my research conducted on MTS and

MARL, attention, namely multi-headed attention via a transformer, proved to be a vital component of the work. And in hindsight, there are several other places within the context of this thesis that attention mechanisms could have been explored. Firstly, a graph attention network could have been used in place the GCN in my ACOPF solution. Secondly, in the world model, MATWM, a transformer could replace the convolutional neural network used in image-based environments. While using attention in these two scenarios may not necessarily improve performance, it highlights the versatility of the mechanism, and represents a promising direction for future work.

Another unifying theme across all three contributions in this thesis is the role of representation learning. Each model is designed not only to solve a task but to learn meaningful internal structures that capture and exploit key relationships in the data, allowing subsequent layers or mechanisms to make better use of the learned representation. In the ACOPF framework, the graph neural network learns embeddings that encode the physical topology and flow characteristics of the power grid, enabling faster solver convergence. In STTRE, the relative spatio-temporal embeddings allow the Transformer to capture latent interactions between variables over time, improving forecasting accuracy. In MATWM, the world model learns a compressed latent representation of the environment's dynamics, which agents use to anticipate future states and improve coordination. These internal representations serve as the foundation for generalization, efficiency, and interpretability. Across all cases, the quality of learned representations proves to be more impactful than simply optimizing for performance metrics, which highlights the importance of understanding the structure of the data you are working with and knowing what tools you have at your disposal for learning representations of said data which allow for optimized representations of that data. This commonality suggests that future work should continue to prioritize architectures that encourage structured and expressive internal representations.

Most importantly, this thesis also reflects my own growth as a researcher. I began solely by applying existing methods to new problems, but gradually advanced toward designing and optimizing entire architectures with many interacting components. Along the way, I engaged deeply with both high-level design choices and low-level details such as loss functions and training stabilizations. This progression not only shaped my final chapter, MATWM, but also marks my development into a more independent and capable researcher, ready to tackle increasingly complex challenges in the field.

Future Directions

Each of the three contributions in this thesis opens up several potential directions for future work, both within their individual problem spaces and at the intersection between them.

In the ACOPF chapter, the GNN-based PPO framework was shown to significantly reduce solver convergence time. One natural extension is to experiment with more expressive graph models, like attention-based GNNs, which might capture dependencies more effectively. Theres also room to explore how both major and minor topology changes affect the performance of the GNN. GNNs are often cited as being robust to changes

in topology, especially in power systems applications, but I pointed out that this claim has limits. While the GNN may generalize to small structural variations, its learned weights are tightly coupled to the relationships between nodes in the training topology. As those relationships change, the relevance of the learned embeddings degrades, and performance can drop off. It would be worth investigating how much topology change the GNN can tolerate before retraining becomes necessary, whether there is a threshold of structural change beyond which performance deteriorates sharply, and whether certain types of changes (e.g., new nodes, line failures, reconfigurations) have more impact than others.

In the STTRE chapter, I proposed a Transformer-based forecasting model with novel relative embeddings, and showed strong improvements in forecasting accuracy. One limitation was scalability the input embedding layer becomes memory-intensive with long sequences and/or large numbers of variables. This is a result of each element having a unique embedding, to not have any instances of sharing weights across variables or timesteps. If embedding layer weights were shared across variables, it would save significant amounts of memory in instances of high variable counts. Another direction of future work is to tailor STTRE to data of a specific domain, allowing it to be better utilized in industry.

For MATWM, I introduced a Transformer-based world model for multi-agent reinforcement learning that supports decentralized rollouts and teammate modeling. While the model performed well on cooperative benchmarks, applying it to competitive or mixed settings is an open challenge. Also, I want to explore new mechanisms to help combat the negative effects of nonstationarity on the decentralized world model, PPO-style training or a recency-biased replay memory sampling structure help with this, but it is likely not enough on their own. Next, I want to expand the teammate predictor to predict information regarding the other agents' states, which could help improve imagined state transitions for the focal agent, and improve the critic. Additionally, MATWM currently uses a CNN for image-based input; replacing it with a Transformer-based encoder like a Vision Transformer (ViT) could improve representation learning and unify the architecture further. Lastly, theres also potential to explore the use of MATWM in real-world multi-agent problems such as robotics or energy systems control.

More broadly, in the non-application-specific contributions, STTRE and MATWM, the goal was to advance areas of deep learning that are still underdeveloped relative to their single-agent or univariate counterparts, and to push them toward being capable of handling more complex, real-world problems. With that in mind, a natural direction for future work is to take these models, which have so far been tested on benchmark datasets, and tailor them toward application-driven use cases that could realistically transfer to industry. This includes adapting STTRE to time series forecasting tasks in energy, finance, or climate modeling, and exploring how MATWM could be used for coordinated control in systems like robotics fleets or distributed energy resources. Translating methods designed for general use to real-world, domain-specific deployment remains a key challenge.

Bibliography

Deep learning. *Nature*, 521(7553):436–444, May 2015. ISSN 0028-0836. doi: 10.1038/nature14539.

M.R. Adaryani and A. Karami. Artificial bee colony algorithm for solving multi-objective optimal power flow problem. *International Journal of Electrical Power & Energy Systems*, 53:219–230, 2013. doi: 10.1016/j.ijepes.2013.04.017.

R.N. Allan, B. Borkowska, and C.H. Grigg. Probabilistic analysis of power flows. *IEEE Transactions on Power Systems*, 6(1):69–76, 1991. doi: 10.1109/59.131005.

S. Aslam, P. P. Aung, A. S. Rafsanjani, et al. Machine learning applications in energy systems: Current trends, challenges, and research directions. *Energy Informatics*, 8:62, 2025. doi: 10.1186/s42162-025-00524-6. URL `https://doi.org/10.1186/s42162-025-00524-6`.

A.-F. Attia, Y.A. Al-Turki, and A.M. Abusorrah. Optimal power flow using adapted genetic algorithm with adjusting population size. *Electric Power Components and Systems*, 40(12):1285–1299, 2012. doi: 10.1080/15325008.2012.689417.

K. Ayan, U. Kiliç, and B. Barakli. Chaotic artificial bee colony algorithm based solution of security and transient stability constrained optimal power flow. *International Journal of Electrical Power & Energy Systems*, 64:136–147, 2014. doi: 10.1016/j.ijepes.2014.07.071.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL `https://arxiv.org/abs/1607.06450`.

Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. Time-series classification with cote: The collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535, 2015. doi: 10.1109/TKDE.2015.2416723.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018. URL `https://arxiv.org/abs/1803.01271`.

K. Baker. Learning warm-start points for ac optimal power flow. In *2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2019a. doi: 10.1109/MLSP.2019.8918695.

Kyri Baker. Learning warm-start points for ac optimal power flow. In *IEEE 29th International Workshop on Machine Learning for Signal Processing*, pages 1–6, 2019b. doi: 10.1109/MLSP.2019.8918690.

Kyri Baker. Solutions of dc opf are never ac feasible. In *Proceedings of the Twelfth ACM International Conference on Future Energy Systems*, page 264268. Association for Computing Machinery, 2021. ISBN 9781450383332. doi: 10.1145/3447555.3464875.

Charles Beattie, Joel Z. Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, Julian Schrittwieser, Keith Anderson, Sarah York, Max Cant, Adam Cain, Adrian Bolton, Stephen Gaffney, Helen King, Demis Hassabis, Shane Legg, and Stig Petersen. Deepmind lab, 2016. URL `https://arxiv.org/abs/1612.03801`.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL `https://arxiv.org/abs/1308.3432`.

P.R. Bijwe and G.K.V. Raju. Fuzzy distribution power flow for weakly meshed systems. *IEEE Transactions on Power Systems*, 21(4):1645–1652, 2006.

B. Borkowska. Probabilistic load flow. *IEEE Transactions on Power Apparatus and Systems*, PAS-93(3):752–759, 1974. doi: 10.1109/TPAS.1974.293972.

G. E. P. Box and David A. Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association*, 65(332):1509–1526, 1970. doi: 10.1080/01621459.1970.10481180. URL `https://www.tandfonline.com/doi/abs/10.1080/01621459.1970.10481180`.

George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time series analysis: forecasting and control*. J. Wiley Sons, Hoboken, N.J, 4 edition, 2008. ISBN 9781118619193.

George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley, 5th edition, 2015.

Leo Breiman. Random forests. *Machine Learning*, 45(1):532, October 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL `https://doi.org/10.1023/A:1010933404324`.

Waqquas A. Bukhsh, Andreas Grothey, Ken I. M. McKinnon, and Paul A. Trodden. Local solutions of the optimal power flow problem. *IEEE Transactions on Power Systems*, 28(4):4780–4788, 2013. doi: 10.1109/TPWRS.2013.2274577.

Raphal Canyasse, Gal Dalal, and Shie Mannor. Supervised learning for optimal power flow as a real-time proxy. In *IEEE Power Energy Society Innovative Smart Grid Technologies Conference*, pages 1–5, 2017. doi: 10.1109/ISGT.2017.8086083.

Yuji Cao, Huan Zhao, Gaoqi Liang, Junhua Zhao, Huanxin Liao, and Chao Yang. Fast and explainable warm-start point learning for ac optimal power flow using decision tree. *International Journal of Electrical Power Energy Systems*, page 109369, 2023. ISSN 0142-0615. doi: https://doi.org/10.1016/j.ijepes.2023.109369.

J. Carpentier. Contribution l'tude du dispatching conomique [contribution to the economic dispatch problem]. *Bulletin de la Socit Franaise des Electriciens*, 3(8):431–447, 1962.

Sneha Chaudhari, Varun Mithal, Gungor Polatkan, and Rohan Ramanath. An attentive survey of attention models. 2021.

Chang Chen, Jaesik Yoon, Yi-Fu Wu, and Sungjin Ahn. Transdreamer: Reinforcement learning with transformer world models. arXiv preprint, under review at ICLR, 2022a. URL https://openreview.net/forum?id=s3K0arSRl4d.

Zekai Chen, Dingshuo Chen, Xiao Zhang, Zixuan Yuan, and Xiuzhen Cheng. Learning graph structures with transformer for multivariate time-series anomaly detection in iot. *IEEE Internet of Things Journal*, 9(12):9179–9189, 2022b. doi: 10.1109/JIOT.2021.3100509.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In Alessandro Moschitti, Bo Pang, and Walter Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179/.

Rich Christie. Power systems test case archive. https://labs.ece.uw.edu/pstca/, 1999. University of Washington.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20 (3):273297, September 1995. ISSN 0885-6125. doi: 10.1023/A:1022627411411. URL https://doi.org/10.1023/A:1022627411411.

Marc-Alexandre Côté, Ákos Kádár, Xingdi Yuan, Ben Kybartas, Tavian Barnes, Emery Fine, James Moore, Ruo Yu Tao, Matthew Hausknecht, Layla El Asri, Mahmoud Adada, Wendy Tay, and Adam Trischler. Textworld: A learning environment for text-based games. *CoRR*, abs/1806.11532, 2018. URL https://arxiv.org/abs/1806.11532.

Suresh Dara, Swetha Dhamercherla, Surender Singh Jadav, C. H. Madhu Babu, and Mohamed Jawed Ahsan. Machine learning in drug discovery: A review. *Artificial Intelligence Review*, 55(3):1947–1999, 2022. doi: 10.1007/s10462-021-10058-4. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC8356896/.

D. Das, S. Ghosh, and D.K. Srinivas. Fuzzy distribution load flow. *International Journal of Electrical Power & Energy Systems*, 21(2):147–151, 1999.

Azad Deihim, Eduardo Alonso, and Dimitra Apostolopoulou. Sttre: A spatio-temporal transformer with relative embeddings for multivariate time series forecasting. *Neural Networks*, 168:549–559, 2023. ISSN 0893-6080. doi: 10.1016/j.neunet.2023.09.039. URL https://doi.org/10.1016/j.neunet.2023.09.039.

Azad Deihim, Dimitra Apostolopoulou, and Eduardo Alonso. Initial estimate of ac optimal power flow with graph neural networks. *Electric Power Systems Research*, 234:110782, 2024. ISSN 0378-7796. doi: 10.1016/j.epsr.2024.110782. URL https://doi.org/10.1016/j.epsr.2024.110782.

Azad Deihim, Eduardo Alonso, and Dimitra Apostolopoulou. Transformer world model for sample efficient multi-agent reinforcement learning, 2025. URL https://arxiv.org/abs/2506.18537.

Marc Peter Deisenroth and Carl Edward Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 465–472, Bellevue, Washington, USA, 2011. Omnipress. ISBN 9781450306195.

Angus Dempster, Francois Petitjean, and Geoffrey I. Webb. Rocket: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):14541495, sep 2020. ISSN 1384-5810. doi: 10.1007/s10618-020-00701-z. URL https://doi.org/10.1007/s10618-020-00701-z.

Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Data Mining and Knowledge Discovery*, KDD '21, page 248257, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467231. URL https://doi.org/10.1145/3447548.3467231.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

F. Diehl. Warm-starting ac optimal power flow with graph neural networks. In *33rd Conference on Neural Information Processing Systems (NeurIPS 2019)*, pages 1–6, 2019.

Jingtao Ding, Yunke Zhang, Yu Shang, Yuheng Zhang, Zefang Zong, Jie Feng, Yuan Yuan, Hongyuan Su, Nian Li, Nicholas Sukiennik, Fengli Xu, and Yong Li. Understanding world or predicting future? a comprehensive survey of world models, 2024. URL https://arxiv.org/abs/2411.14499.

Hermann W. Dommel and William F. Tinney. Optimal power flow solutions. *IEEE Transactions on Power Apparatus and Systems*, PAS-87(10):1866–1876, 1968. doi: 10.1109/TPAS.1968.292150.

Priya L. Donti, David Rolnick, and J. Zico Kolter. DC3: A learning method for optimization with hard constraints. *CoRR*, 2021.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Ziheng Duan, Haoyan Xu, Yueyang Wang, Yida Huang, Anni Ren, Zhongbin Xu, Yizhou Sun, and Wei Wang. Multivariate time-series classification with hierarchical variational graph pooling. *Neural Networks*, 154:481–490, 2022. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2022.07.032. URL https://www.sciencedirect.com/science/article/pii/S0893608022002970.

Selcuk Duman, Ugur Guvenc, Yusuf Sonmez, and Nezih Yorukeren. Optimal power flow using gravitational search algorithm. *Energy Conversion and Management*, 59:86–95, 2012.

Vladimir Egorov and Alexei Shpilman. Scalable multi-agent model-based reinforcement learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, AAMAS '22, pages 381–390, Richland, SC, 2022. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 9781450392136.

Attia El-Fergany and Hany M. Hasanien. Single and multi-objective optimal power flow using grey wolf optimizer and differential evolution algorithms. *Electric Power Components and Systems*, 43(13):1548–1559, 2015.

Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. InceptionTime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery*, 34(6):1936–1962, sep 2020. doi: 10.1007/s10618-020-00710-y. URL `https://doi.org/10.10072Fs10618-020-00710-y`.

Tuo Feng, Wenguan Wang, and Yi Yang. A survey of world models for autonomous driving, 2025. URL `https://arxiv.org/abs/2501.11260`.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018. doi: 10.1609/aaai.v32i1.11794. URL `https://ojs.aaai.org/index.php/AAAI/article/view/11794`.

Claude Formanek, Asad Jeewa, Jonathan Shock, and Arnu Pretorius. Off-the-grid marl: Datasets and baselines for offline multi-agent reinforcement learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, AAMAS '23, pages 2442–2444, London, United Kingdom, 2023. International Foundation for Autonomous Agents and Multiagent Systems.

Stephen Frank, Ingrida Steponavice, and Steffen Rebennack. Optimal power flow: a bibliographic survey i: Formulations and deterministic methods. *Energy Systems*, 3 (3):221–258, 2012. doi: 10.1007/s12667-012-0056-y.

M. Gao, J. Yu, Z. Yang, and J. Zhao. A physics-guided graph convolution neural network for optimal power flow. *IEEE Transactions on Power Systems*, 2023.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey Gordon, David Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL `https://proceedings.mlr.press/v15/glorot11a.html`.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27, 2014.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Jake Grigsby, Zhe Wang, and Yanjun Qi. Long-range transformers for dynamic spatiotemporal forecasting. *CoRR*, abs/2109.12218, 2021. URL https://arxiv.org/abs/2109.12218.

William H. Guss, Brandon Houghton, Nicholay Topin, Akash Velu, Sam Codel, Milani Alfredo, Shariq Trivedi Iqbal, Dipam Dey, Steven Wong, Kalpesh Gopalakrishnan, et al. Minerl: A large-scale dataset of minecraft demonstrations. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019.

David Ha and Jürgen Schmidhuber. World models. *Zenodo*, 2018. doi: 10.5281/ZENODO.1207631. URL https://zenodo.org/record/1207631.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2019. URL https://arxiv.org/abs/1912.01603.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models, 2020. URL https://arxiv.org/abs/2010.02193.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2024. URL https://arxiv.org/abs/2301.04104.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 10251035, 2017. ISBN 9781510860964.

Dong Han, Beni Mulyana, Vladimir Stankovic, and Samuel Cheng. A survey on deep reinforcement learning algorithms for robotic manipulation. *Sensors*, 23(7):3762, 2023a. ISSN 1424-8220. doi: 10.3390/s23073762. URL https://www.mdpi.com/1424-8220/23/7/3762.

Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, Zhaohui Yang, Yiman Zhang, and Dacheng Tao. A survey on vision transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):87–110, 2023b. doi: 10.1109/TPAMI.2022.3152247.

X. Han, C. Mu, J. Yan, and Z. Niu. An autonomous control technology based on deep reinforcement learning for optimal active power dispatch. *International Journal of Electrical Power & Energy Systems*, 145:108686, 2023c.

J. B. Hansen, S. N. Anfinsen, and F. M. Bianchi. Power flow balancing with decentralized graph neural networks. *IEEE Transactions on Power Systems*, 2022.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, New York, NY, 2 edition, 2009. ISBN 978-0-387-84858-7. doi: 10.1007/978-0-387-84858-7. URL https://link.springer.com/book/10.1007/978-0-387-84858-7.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33:750–797, 2019.

Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. A very condensed survey and critique of multiagent deep reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, AAMAS '20, pages 2146–2148, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Ian A. Hiskens. Stability of limit cycles in hybrid systems. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, page 6. IEEE, 2001. doi: 10.1109/HICSS.2001.926280.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):17351780, nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

A.L. Hossain, M. Rokonuzzaman, N. Amin, J. Zhang, M.K. Mishu, W.-S. Tan, M.R. Islam, and R.B. Roy. Probabilistic load flow–based optimal placement and sizing of distributed generators. In *Proceedings of the 2019 4th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT)*, pages 1067–1072, Mysuru, India, December 2019.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, and Douglas Eck. An improved relative self-attention mechanism for transformer with application to music generation. *CoRR*, abs/1809.04281, 2018. URL `http://arxiv.org/abs/1809.04281`.

Shengyi Huang, Santiago Ontañón, Chris Bamford, and Lukasz Grela. Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning. *arXiv*, 2105.13807, 2021. URL `https://arxiv.org/abs/2105.13807`.

Justus A. Ilemobayo, Olamide Durodola, Oreoluwa Alade, Opeyemi J. Awotunde, Adewumi T. Olanrewaju, Olumide Falana, Adedolapo Ogungbire, Abraham Osinuga, Dabira Ogunbiyi, Ark Ifeanyi, Ikenna E. Odezuligbo, and Oluwagbotemi E. Edu. Hyperparameter tuning in machine learning: A comprehensive review. *Journal of Engineering Research and Reports*, 26(6):388–395, 2024. doi: 10.9734/JERR/2024/v26i61188. URL `https://journaljerr.com/index.php/JERR/article/view/1188`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015. URL `https://arxiv.org/abs/1502.03167`.

Andrew Jaegle, Felix Gimeno, Andy Brock, Oriol Vinyals, Andrew Zisserman, and Joao Carreira. Perceiver: General perception with iterative attention. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4651–4664. PMLR, 18–24 Jul 2021.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017. URL `https://arxiv.org/abs/1611.01144`.

Y. Jia, X. Bai, L. Zheng, Z. Weng, and Y. Li. Convopf-dop: A data-driven method for solving ac-opf based on cnn considering different operation patterns. *IEEE Transactions on Power Systems*, 38(1):853–860, 2023.

S. Kahourzade, A. Mahmoudi, and H.B. Mokhlis. A comparative study of multi-objective optimal power based on particle swarm, evolutionary programming and genetic algorithm. *Electrical Engineering*, 97(1):1–12, 2015.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H. Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020. URL `https://openreview.net/forum?id=S1xCPJHtDB`.

B.M. Kalesar and A.R. Sei. Fuzzy load flow in balanced and unbalanced radial distribution systems incorporating composite load model. *International Journal of Electrical Power & Energy Systems*, 32(1):17–23, 2010.

N. Kalfallah, F. Benzergua, I. Cherki, and A. Chaker. Use of genetic algorithm and particle swarm optimisation methods for the optimal control of reactive power in western algerian power system. *Energy Procedia*, 74:265–272, 2015.

Christos Kaplanis, Murray Shanahan, and Claudia Clopath. Policy consolidation for continual reinforcement learning, 2019. URL `https://arxiv.org/abs/1902.00255`.

Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. Lstm fully convolutional networks for time series classification. *IEEE Access*, 6:1662–1669, 2018. doi: 10.1109/ACCESS.2017.2779939.

Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 116:237–245, 2019. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2019.04.014. URL `https://www.sciencedirect.com/science/article/pii/S0893608019301200`.

Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus A. Brubaker. Time2vec: Learning a vector representation of time. *CoRR*, abs/1907.05321, 2019. URL `http://arxiv.org/abs/1907.05321`.

Hooman Khaloie, Mihly Dolnyi, Jean-Franois Toubeau, and Franois Valle. Review of machine learning techniques for optimal power flow. *Applied Energy*, 388:125637, 2025. ISSN 0306-2619. doi: 10.1016/j.apenergy.2025.125637. URL https://www.sciencedirect.com/science/article/pii/S0306261925003678.

H. Khazaei and Y. Zhao. Physics-aware fast learning and inference for predicting active set of dc-opf. In *2022 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, pages 1–5. IEEE, 2022. doi: 10.1109/ISGT50606.2022.9817503.

Jung In Kim, Young Jae Lee, Jongkook Heo, Jinhyeok Park, Jaehoon Kim, Sae Rin Lim, Jinyong Jeong, and Seoung Bum Kim. Sample-efficient multi-agent reinforcement learning with masked reconstruction. *PLOS ONE*, 18(9):e0291545, 2023.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes, 2013. URL https://arxiv.org/abs/1312.6114.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, 2016.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

Sampo Kuutti, Richard Bowden, Yaochu Jin, Phil Barber, and Saber Fallah. A survey of deep learning applications to autonomous vehicle control. *CoRR*, abs/1912.10773, 2019. URL http://arxiv.org/abs/1912.10773.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. Openspiel: A framework for reinforcement learning in games. *CoRR*, abs/1908.09453, 2019. doi: 10.48550/arXiv.1908.09453. URL https://arxiv.org/abs/1908.09453.

Pablo Lara-Bentez, Manuel Carranza-Garca, Daniel Lara-Bentez, and Francisco Herrera. An experimental review on deep learning architectures for time series forecasting. *International Journal of Neural Systems*, 31(03):2130003, 2021.

Javad Lavaei and Steven H. Low. Zero duality gap in optimal power flow problem. *IEEE Transactions on Power Systems*, 27(1):92–107, 2012. doi: 10.1109/TPWRS.2011.2160974.

Yann LeCun, Bernhard Boser, John S. Denker, Donnie Henderson, Richard E. Howard, Wayne Hubbard, and Lawrence D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. doi: 10.1162/neco.1989.1.4.541. URL https://doi.org/10.1162/neco.1989.1.4.541.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015. doi: 10.1038/nature14539. URL https://www.nature.com/articles/nature14539.

Kyungeun Lee, Moonjung Eo, Euna Jung, Yoonjin Yoon, and Wonjong Rhee. Short-term traffic prediction with deep neural networks: A survey. *IEEE Access*, 9:54739–54756, 2021. doi: 10.1109/ACCESS.2021.3071174.

Jianjun Lei, Xiangwei Zhu, and Ying Wang. Bat: Block and token self-attention for speech emotion recognition. *Neural Networks*, 156:67–80, 2022. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2022.09.022. URL https://www.sciencedirect.com/science/article/pii/S0893608022003720.

Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Neural Information Processing Systems*, page 52435253, 2019. doi: 10.5555/3454287.3454758. URL https://doi/10.5555/3454287.3454758.

Y. Li, C. Zhao, and C. Liu. Solving non-linear optimization problem in engineering by model-informed generative adversarial network (mi-gan). In *2022 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 198–205. IEEE, 2022. doi: 10.1109/ICDMW58026.2022.00035.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200209, 2021. doi: 10.1098/rsta.2020.0209. URL https://royalsocietypublishing.org/doi/10.1098/rsta.2020.0209.

Ray Lincoln and Ray D. Zimmerman. Pypower: Power flow and optimal power flow in python. https://github.com/rwl/PYPOWER, 2011. Python port of MATPOWER.

Jason Lines, Sarah Taylor, and Anthony Bagnall. Time series classification with hive-cote: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data*, 12(5), jul 2018. ISSN 1556-4681. doi: 10.1145/3182382. URL https://doi.org/10.1145/3182382.

Minghao Liu, Shengqi Ren, Siyuan Ma, Jiahui Jiao, Yizhou Chen, Zhiguang Wang, and Wei Song. Gated transformer networks for multivariate time series classification. *CoRR*, abs/2103.14438, 2021a. URL https://arxiv.org/abs/2103.14438.

Qihan Liu, Jianing Ye, Xiaoteng Ma, Jun Yang, Bin Liang, and Chongjie Zhang. Efficient multi-agent reinforcement learning by planning. In *Proceedings of the Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=CpnKq3UJwp.

S. Liu, Y. Guo, W. Tang, H. Sun, and W. Huang. Predicting active constraints set in security-constrained optimal power flow via deep neural network. In *2021 IEEE Power & Energy Society General Meeting (PESGM)*, pages 01–05. IEEE, 2021b. doi: 10.1109/PESGM46819.2021.9637964.

S. Liu, C. Wu, and H. Zhu. Graph neural networks for learning real-time prices in electricity market. *arXiv preprint*, 2021c.

S. Liu, Y. Guo, W. Tang, H. Sun, W. Huang, and J. Hou. Varying condition scopf optimization based on deep learning and knowledge graph. *IEEE Transactions on Power Systems*, 2022a. doi: 10.1109/TPWRS.2022.3195351.

S. Liu, C. Wu, and H. Zhu. Topology-aware graph neural networks for learning feasible and adaptive ac-opf solutions. *IEEE Transactions on Power Systems*, 2022b.

Shaohui Liu, Chengyang Wu, and Hao Zhu. Topology-aware graph neural networks for learning feasible and adaptive ac-opf solutions. *IEEE Transactions on Power Systems*, pages 1–11, 2022c. doi: 10.1109/TPWRS.2022.3230555.

Shizhan Liu, Hang Yu, Cong Liao, Jianguo Li, Weiyao Lin, Alex X. Liu, and Schahram Dustdar. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. In *International Conference on Learning Representations*, 2022d. URL `https://openreview.net/forum?id=0EXmFzUn5I`.

Sicong Liu, Xi Sheryl Zhang, Yushuo Li, Yifan Zhang, and Jian Cheng. On the data-efficiency with contrastive image transformation in reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023.

Xin Liu, Yaran Chen, Haoran Li, and Dongbin Zhao. Learning future representation with synthetic observations for sample-efficient reinforcement learning. *Science China Information Sciences*, 68(5):150202, 2025.

G. López-Cardona, P. Bernárdez, P. Barlet-Ros, and A. Cabellos-Aparicio. Proximal policy optimization with graph neural networks for optimal power flow. *arXiv preprint arXiv:2212.12470*, 2022. URL `https://arxiv.org/abs/2212.12470`.

Steven H. Low. Convex relaxation of optimal power flow—part i: Formulations and equivalence. *IEEE Transactions on Control of Network Systems*, 1(1):15–27, 2014. doi: 10.1109/TCNS.2014.2309732.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, NeurIPS '17, pages 6382–6393, Red Hook, NY, USA, 2017. Curran Associates Inc.

Chujie Lu, Sihui Li, and Zhengjun Lu. Building energy prediction using artificial neural networks: A literature survey. *Energy and Buildings*, 262:111718, 2022. ISSN 0378-7788. doi: https://doi.org/10.1016/j.enbuild.2021.111718. URL `https://www.sciencedirect.com/science/article/pii/S0378778821010021`.

Fan-Ming Luo, Tian Xu, Hang Lai, Xiong-Hui Chen, Weinan Zhang, and Yang Yu. A survey on model-based reinforcement learning. *Science China Information Sciences*, 67(2):121101, 2024.

Guozheng Ma, Linrui Zhang, Haoyu Wang, Lu Li, Zilin Wang, Zhen Wang, Li Shen, Xueqian Wang, and Dacheng Tao. Learning better with less: Effective augmentation for sample-efficient visual reinforcement learning. In *Proceedings of the Thirty-Seventh Conference on Neural Information Processing Systems (NeurIPS)*, 2023. URL `https://openreview.net/forum?id=jRL6ErxMVB`.

Zheng Ma, Junyu Xuan, Yu Guang Wang, Ming Li, and Pietro Liò. Path integral based convolution and pooling for graph neural networks. In *Advances in Neural Information Processing Systems*, pages 16421–16433, 2020.

Elena Merdjanovska and Aleksandra Rashkovska. Comprehensive survey of computational ecg analysis: Databases, methods and applications. *Expert Systems with Applications: An International Journal*, 203(C), oct 2022. ISSN 0957-4174. doi: 10.1016/j.eswa.2022.117206. URL `https://doi.org/10.1016/j.eswa.2022.117206`.

Edan Jacob Meyer, Adam White, and Marlos C. Machado. Harnessing discrete representations for continual reinforcement learning. *Reinforcement Learning Journal*, 2: 606–628, 2024.

Vincent Micheli, Eloi Alonso, and Franois Fleuret. Transformers are sample-efficient world models. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL `https://openreview.net/forum?id=vhFu1Acb0xb`.

Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110:32113243, December 2021. ISSN 0885-6125. doi: 10.1007/s10994-021-06057-9.

Ebikella Mienye, Nobert Jere, George Obaido, Ibomoiye Domor Mienye, and Kehinde Aruleba. Deep learning in finance: A survey of applications and techniques. *AI*, 5(4): 2066–2091, 2024. ISSN 2673-2688. doi: 10.3390/ai5040101. URL `https://www.mdpi.com/2673-2688/5/4/101`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16 (1):1–118, 2023. ISSN 1935-8237. doi: 10.1561/2200000086.

Erfan Mohagheghi, Mansour Alramlawi, Aouss Gabash, and Pu Li. A survey of real-time optimal power flow. *Energies*, 2018. ISSN 1996-1073. doi: 10.3390/en11113142.

Mohamed Hussien Moharam, Omar Hany, Ahmed Hany, Amenah Mahmoud, Mariam Mohamed, and Sohila Saeed. Anomaly detection using machine learning and adopted digital twin concepts in radio environments. *Scientific Reports*, 15:18352, 2025. doi: 10.1038/s41598-025-02759-5. URL `https://www.nature.com/articles/s41598-025-02759-5`.

James A. Momoh, Mohamed E. El-Hawary, and Ram Adapa. A review of selected optimal power flow literature to 1993. i. nonlinear and quadratic programming approaches. *IEEE Transactions on Power Systems*, 14(1):96–104, 1999. doi: 10.1109/59.744492.

Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence*, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014602.

Rasananda Muduli, Debashisha Jena, and Tukaram Moger. A survey on load frequency control using reinforcement learning-based data-driven controller. *Applied Soft Computing*, 166:112203, 2024. doi: 10.1016/j.asoc.2024.112203. URL `https://www.sciencedirect.com/science/article/abs/pii/S1568494624009773`.

Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, 2018. doi: 10.1109/ICRA.2018.8463189.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*, pages 807–814, Haifa, Israel, 2010. Omnipress. ISBN 9781605589077.

MohammadReza Nazari, Afshin Oroojlooy, Martin Takáč, and Lawrence V. Snyder. Reinforcement learning for solving the vehicle routing problem. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 9861–9871, Red Hook, NY, USA, 2018. Curran Associates, Inc. doi: 10.5555/3327546.3327651. URL `https://papers.nips.cc/paper/8190-reinforcement-learning-for-solving-the-vehicle-routing-problem`.

Daniel Neimark, Omri Bar, Maya Zohar, and Dotan Asselmann. Video transformer network. *Proceedings of the IEEE/CVF International Conference on Computer Vision.*, abs/2102.00719, 2021.

Rahul Nellikkath and Spyros Chatzivasileiadis. Physics-informed neural networks for ac optimal power flow. *Electric Power Systems Research*, 2022. ISSN 0378-7796. doi: 10.1016/j.epsr.2022.108412.

H. Nie, Y. Chen, Y. Song, and S. Huang. A general real-time opf algorithm using ddpg with multiple simulation platforms. In *2019 IEEE Innovative Smart Grid Technologies-Asia (ISGT Asia)*, pages 3713–3718. IEEE, 2019.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2 edition, 2006. doi: 10.1007/978-0-387-40065-5.

Office for National Statistics. Business insights and impact on the uk economy: 2 october 2025, October 2025. URL https://www.ons.gov.uk/businessindustryandtrade/business/businessservices/bulletins/businessinsightsandimpactontheukeconomy/2october2025. Accessed: 2026-04-29.

Xiang Pan, Minghua Chen, Tianyu Zhao, and Steven H. Low. Deepopf: A feasibility-optimized deep neural network approach for ac optimal power flow problems. *IEEE Systems Journal*, pages 673–683, 2023. doi: 10.1109/JSYST.2022.3201041.

George Papadopoulos, Andreas Kontogiannis, Foteini Papadopoulou, Chaido Poulianou, Ioannis Koumentis, and George Vouros. An extended benchmarking of multi-agent reinforcement learning algorithms in complex fully cooperative tasks, 2025. URL https://arxiv.org/abs/2502.04773.

G. Papaefthymiou, P. Schavemaker, L. van der Sluis, W. Kling, D. Kurowicka, and R. Cooke. Integration of stochastic generation in power systems. *International Journal of Electrical Power & Energy Systems*, 28(9):655–667, 2006. doi: 10.1016/j.ijepes.2006.03.003.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. URL https://openreview.net/forum?id=cIrPX-Sn5n.

Nisarg P. Patel, Raj Parekh, Nihar Thakkar, Rajesh Gupta, Sudeep Tanwar, Gulshan Sharma, Innocent E. Davidson, and Ravi Sharma. Fusion in cryptocurrency price prediction: A decade survey on recent advancements, architecture, and potential future directions. *IEEE Access*, 10:34511–34538, 2022. doi: 10.1109/ACCESS.2022.3163023.

Tran Phuoc, Pham Thi Kim Anh, Phan Huy Tam, and Chien V. Nguyen. Applying machine learning algorithms to predict the stock price trend in the stock market: The case of vietnam. *Humanities and Social Sciences Communications*, 11(1):393, 2024. doi: 10.1057/s41599-024-02807-x. URL https://www.nature.com/articles/s41599-024-02807-x.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. ISSN 0041-5553. doi: 10.1016/0041-5553(64)90137-5. URL https://www.sciencedirect.com/science/article/pii/0041555364901375.

Xinyuan Qi, Kai Hou, Tong Liu, Zhongzhong Yu, Sihao Hu, and Wenwu Ou. From known to unknown: Knowledge-guided transformer for time-series sales forecasting in alibaba. *CoRR*, abs/2109.08381, 2021. URL https://arxiv.org/abs/2109.08381.

Mingcheng Qu, Ganlin Deng, Donglin Di, Jianxun Cui, and Tonghua Su. Dual attentional transformer for video visual relation prediction. *Neurocomputing*, 550:126372, 2023. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.126372.

Elahe Rahimian, Soheil Zabihi, Seyed Farokh Atashzar, Amir Asif, and Arash Mohammadi. Xceptiontime: A novel deep architecture based on depthwise separable

convolutions for hand gesture classification. *CoRR*, abs/1911.03803, 2019. URL http://arxiv.org/abs/1911.03803.

J. Rahman, C. Feng, and J. Zhang. A learning-augmented approach for ac optimal power flow. *International Journal of Electrical Power & Energy Systems*, 130:106908, 2021.

N. L. Rane, S. K. Mallick, O. Kaya, and J. Rane. Applications of deep learning in healthcare, finance, agriculture, retail, energy, manufacturing, and transportation: A review. In *Applied Machine Learning and Deep Learning: Architectures and Techniques*, pages 132–152. Deep Science Publishing, 2024. doi: 10.70593/978-81-981271-4-3_7. URL https://doi.org/10.70593/978-81-981271-4-3_7.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21 (1):178:1–178:51, 2020. ISSN 1532-4435.

Benedetto-Giuseppe Risi, Francesco Riganti-Fulginei, and Antonino Laudani. Modern techniques for the optimal power flow problem: State of the art. *Energies*, 15(17): 6387, 2022. ISSN 1996-1073. doi: 10.3390/en15176387. URL https://www.mdpi.com/1996-1073/15/17/6387.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, September 1951. doi: 10.1214/aoms/1177729586. URL https://doi.org/10.1214/aoms/1177729586.

Jan Robine, Marc Hftmann, Tobias Uelwer, and Stefan Harmeling. Transformer-based world models are happy with 100k interactions. In *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=TdBaDGCpjly.

Sergio Rozada, Dimitra Apostolopoulou, and Eduardo Alonso. A deep multi-agent reinforcement learning approach to autonomous load frequency control. In *Proceedings of the 2020 IEEE Power & Energy Society General Meeting*, 2020. doi: 10.1109/PESGM41954.2020.9281614. URL https://openaccess.city.ac.uk/id/eprint/23763/.

Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: A review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401449, mar 2021. ISSN 1384-5810. doi: 10.1007/s10618-020-00727-3. URL https://doi.org/10.1007/s10618-020-00727-3.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart, James L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*, pages 318–362, Cambridge, MA, USA, 1986a. MIT Press. URL https://api.semanticscholar.org/CorpusID:62245742.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986b. doi: 10.1038/323533a0. URL https://doi.org/10.1038/323533a0.

Gavin A. Rummery and Mahesan Niranjan. On-line q-learning using connectionist systems. Technical report, University of Cambridge, Department of Engineering, 1994.

S. A. Sadat and M. Sahraei-Ardakani. Initializing successive linear programming solver for acopf using machine learning. In *2020 52nd North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2021. doi: 10.1109/NAPS50074.2021.9449706.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, AAMAS '19, pages 2186–2188, Montreal, QC, Canada, 2019. International Foundation for Autonomous Agents and Multiagent Systems.

A. R. Sayed, C. Wang, H. Anis, and T. Bi. Feasibility constrained online calculation for real-time optimal power flow: A convex constrained deep reinforcement learning approach. *IEEE Transactions on Power Systems*, 2022.

A. R. Sayed, X. Zhang, G. Wang, C. Wang, and J. Qiu. Optimal operable power flow: Sample-efficient holomorphic embedding-based reinforcement learning. *IEEE Transactions on Power Systems*, pages 1–13, 2023.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588:604–609, December 2020. doi: 10.1038/s41586-020-03051-4.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, 2017.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *CoRR*, abs/1803.02155, 2018. URL http://arxiv.org/abs/1803.02155.

Sheng Shen, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. Powernorm: Rethinking batch normalization in transformers. ICML'20. JMLR.org, 2020. doi: https://doi.org/10.1007/s10994-021-06057-9.

Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I. Webb. Ts-chief: A scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery*, 34(3):742775, may 2020. ISSN 1384-5810. doi: 10.1007/s10618-020-00679-8. URL https://doi.org/10.1007/s10618-020-00679-8.

Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:60, 2019.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 387–395, 2014.

Andrea Silvestrini and David Veredas. Temporal aggregation of univariate and multivariate time series models: A survey. *Journal of Economic Surveys*, 22(3):458–497, 2008. doi: https://doi.org/10.1111/j.1467-6419.2007.00538.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-6419.2007.00538.x`.

P. Somasundaram, K. Kuppusamy, and R.P.K. Devi. Evolutionary programming based security constrained optimal power flow. *Electric Power Systems Research*, 72(2):137–145, 2004.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 5887–5896. PMLR, 2019.

Ching-Chang Su. Probabilistic load-flow computation using point estimate method. *IEEE Transactions on Power Systems*, 20(4):1843–1851, 2005. doi: 10.1109/TPWRS.2005.857982.

Junjie Sun and Leigh Tesfatsion. Dc optimal power flow formulation and solution using quadprogj. 04 2006.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, AAMAS '18, pages 2085–2087, Stockholm, Sweden, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2(4):160–163, July 1991. ISSN 0163-5719. doi: 10.1145/122344.122377.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, and Praveen Ravi. Pettingzoo: A standard api for multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NeurIPS)*, NeurIPS '21, pages 1152–1163, Red Hook, NY, USA, 2021. Curran Associates Inc.

Edan Toledo. Codreamer: Communication-based decentralised world models. In *Coordination and Cooperation for Multi-Agent Reinforcement Learning Methods Workshop*, 2024. URL https://openreview.net/forum?id=f2bgGy7Af7.

Marcos Treviso, Ji-Ung Lee, Tianchu Ji, Betty van Aken, Qingqing Cao, Manuel R. Ciosici, Michael Hassid, Kenneth Heafield, Sara Hooker, Colin Raffel, Pedro H. Martins, André F. T. Martins, Jessica Zosa Forde, Peter Milder, Edwin Simpson, Noam Slonim, Jesse Dodge, Emma Strubell, Niranjan Balasubramanian, Leon Derczynski, Iryna Gurevych, and Roy Schwartz. Efficient methods for natural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 11:826–860, 2023. doi: 10.1162/tacl_a_00577. URL https://doi.org/10.1162/tacl_a_00577.

Maria Trigka and Elias Dritsas. A comprehensive survey of deep learning approaches in image processing. *Sensors*, 25(2):531, 2025. ISSN 1424-8220. doi: 10.3390/s25020531. URL https://www.mdpi.com/1424-8220/25/2/531.

Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS)*, NeurIPS'17, pages 6309–6318, Long Beach, California, USA, 2017. Curran Associates Inc.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30 (1), 2016.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

M. Varadarajan and K.S. Swarup. Differential evolutionary algorithm for optimal reactive power dispatch. *International Journal of Electrical Power & Energy Systems*, 30(8): 435–441, 2008.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Petar Velikovi, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Li, and Yoshua Bengio. Graph attention networks. *CoRR*, 2018.

J. Wang and P. Srikantha. Fast optimal power flow with guarantees via an unsupervised generative model. *IEEE Transactions on Power Systems*, 2022. doi: 10.1109/TPWRS.2022.3214567.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex dueling multi-agent q-learning. In *Proceedings of the 9th International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=RcmkOxxIQV.

T. Wang and Y. Tang. Transfer-reinforcement-learning-based rescheduling of differential power grids considering security constraints. *Applied Energy*, 306:118121, 2022.

Zhenqi Wang, Jan-Hendrik Menke, Florian Schfer, Martin Braun, and Alexander Scheidler. Approximating multi-purpose ac optimal power flow with reinforcement trained artificial neural network, 2022.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1995–2003, 2016.

Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4): 279–292, 1992. doi: 10.1007/BF00992698.

Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey, 2022. URL `https://arxiv.org/abs/2202.07125`.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. doi: 10.1007/BF00992696.

Alec Wilson, William Holmes, Ryan Menzies, and Kez Smithson Whitehead. Applying action masking and curriculum learning techniques to improve data efficiency and overall performance in operational technology cyber security using reinforcement learning, 2024. URL `https://arxiv.org/abs/2409.10563`.

J. H. Woo, L. Wu, J.-B. Park, and J. H. Roh. Real-time optimal power flow using twin delayed deep deterministic policy gradient algorithm. *IEEE Access*, 8:213611–213618, 2020.

Allen J. Wood, Bruce F. Wollenberg, and Gerald B. Sheblé. *Power Generation, Operation, and Control*. John Wiley & Sons, USA, 2013.

H. Wu, M. Wang, Z. Xu, and Y. Jia. Active constraint identification assisted dc optimal power flow. In *2022 IEEE/IAS Industrial and Commercial Power System Asia (ICPS Asia)*, pages 185–189. IEEE, 2022. doi: 10.1109/ICPSAsia55496.2022.9949655.

Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. In *Advances in Neural Information Processing Systems*, volume 34, pages 22419–22430. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/bcc0d400288793e8bdcd7c19a8ac0c2b-Paper.pdf`.

Sifan Wu, Xi Xiao, Qianggang Ding, Peilin Zhao, Ying Wei, and Junzhou Huang. Adversarial sparse transformer for time series forecasting. In *Advances in Neural Information Processing Systems*, volume 33, pages 17105–17115. Curran Associates, Inc., 2020a. URL `https://proceedings.neurips.cc/paper/2020/file/c6b8c8d762da15fa8dbbdfb6baf9e260-Paper.pdf`.

T. Wu, A. Scaglione, and D. Arnold. Constrained reinforcement learning for stochastic dynamic optimal power flow control. In *2023 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5. IEEE, 2023.

Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Data Mining and Knowledge Discovery*, KDD '20, page 753763, New York, NY, USA, 2020b. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403118. URL https://doi.org/10.1145/3394486.3403118.

T. Xiao, Y. Chen, H. Diao, S. Huang, and C. Shen. On fast-converged reinforcement learning for optimal dispatch of large-scale power systems under transient security constraints. *arXiv preprint arXiv:2304.08320*, 2023.

Zhiwei Xu, Bin Zhang, Yuan Zhan, Yunpeng Baiia, Guoliang Fan, et al. Mingling foresight with imagination: Model-based cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pages 11327–11340, 2022.

Yahoo!Finance, 06 2022. URL https://finance.yahoo.com/quote/UBER/history?p=UBER.

Z. Yan and Y. Xu. Real-time optimal power flow: A lagrangian based deep reinforcement learning approach. *IEEE Transactions on Power Systems*, 35(4):3270–3273, 2020.

K. Yang, W. Gao, and R. Fan. Optimal power flow estimation using one-dimensional convolutional neural network. In *2021 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2021.

T. Yang, L. Zhao, W. Li, and Albert Y. Zomaya. Reinforcement learning in sustainable energy and electric systems: A survey. *Annual Reviews in Control*, 49:145–163, 2020.

Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. Mastering atari games with limited data. In *Advances in Neural Information Processing Systems*, volume 34, pages 25476–25488. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/d5eca8dc3820cad9fe56a3bafda65ca1-Paper.pdf.

Xi Ye and Guillaume-Alexandre Bilodeau. Video prediction by efficient transformers. *Image and Vision Computing*, 130:104612, 2023. ISSN 0262-8856. doi: https://doi.org/10.1016/j.imavis.2022.104612.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. In *Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS)*, NeurIPS '22', page 1787, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

L. Yu, S. Qin, M. Zhang, C. Shen, T. Jiang, and X. Guan. A review of deep reinforcement learning for smart building energy management. *IEEE Internet of Things Journal*, 8 (15):12046–12063, 2021.

George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. A transformer-based framework for multivariate time series representation learning. In *Data Mining and Knowledge Discovery*, KDD '21, page 21142124, New York, NY, USA, 2021. Association for Computing Machinery. ISBN

9781450383325. doi: 10.1145/3447548.3467401. URL https://doi.org/10.1145/3447548.3467401.

Hongxin Zhang, Zeyuan Wang, Qiushi Lyu, Zheyuan Zhang, Sunli Chen, Tianmin Shu, Behzad Dariush, Kwonjoon Lee, Yilun Du, and Chuang Gan. COMBO: Compositional world models for embodied multi-agent cooperation. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=YXRyYkb1im.

Ling Zhang and Baosen Zhang. Learning to solve the ac optimal power flow via a lagrangian approach. In *North American Power Symposium*, pages 1–6, 2022. doi: 10.1109/NAPS56150.2022.10012237.

Weipu Zhang, Gang Wang, Jian Sun, Yetian Yuan, and Gao Huang. Storm: Efficient stochastic transformer based world models for reinforcement learning. *Advances in Neural Information Processing Systems*, 36:27147–27166, 2023.

Yang Zhang, Chenjia Bai, Bin Zhao, Junchi Yan, Xiu Li, and Xuelong Li. Decentralized transformers with centralized aggregation are sample-efficient multi-agent world models, 2024. URL https://arxiv.org/abs/2406.15836.

Shixin Zhao, Feng Pan, Anni Jiang, Hao Zhang, and Qiuqi Gao. The unmanned vehicle on-ramp merging model based on AM-MAPPO algorithm. *Scientific Reports*, 14(1), 2024. doi: 10.1038/s41598-024-70509-0. URL https://www.nature.com/articles/s41598-024-70509-0.

H. Zhen, H. Zhai, W. Ma, L. Zhao, Y. Weng, Y. Xu, J. Shi, and X. He. Design and tests of reinforcement-learning-based optimal power flow solution generator. *Energy Reports*, 8:43–50, 2022.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *Association for the Advancement of Artificial Intelligence*, 35:11106–11115, May 2021. doi: 10.1609/aaai.v35i12.17325. URL https://ojs.aaai.org/index.php/AAAI/article/view/17325.

M. Zhou, M. Chen, and S. H. Low. Deepopf-ft: One deep neural network for multiple ac-opf problems with flexible topology. *IEEE Transactions on Power Systems*, 38(1): 964–967, 2023.

Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting. volume 162 of *Proceedings of Machine Learning Research*, pages 27268–27286, 17–23 Jul 2022a. URL https://proceedings.mlr.press/v162/zhou22g.html.

Y. Zhou, B. Zhang, C. Xu, T. Lan, R. Diao, D. Shi, Z. Wang, and W.-J. Lee. A data-driven method for fast ac optimal power flow solutions via deep reinforcement learning. *Journal of Modern Power Systems and Clean Energy*, 8(6):1128–1139, 2020a. doi: 10.35833/MPCE.2020.000249.

Y. Zhou, W.-J. Lee, R. Diao, and D. Shi. Deep reinforcement learning based real-time ac optimal power flow considering uncertainties. *Journal of Modern Power Systems and Clean Energy*, 10(5):1098–1109, 2022b. doi: 10.35833/MPCE.2021.000293.

Yuhao Zhou, Bei Zhang, Chunlei Xu, Tu Lan, Ruisheng Diao, Di Shi, Zhiwei Wang, and Wei-Jen Lee. A data-driven method for fast ac optimal power flow solutions via deep reinforcement learning. *Journal of Modern Power Systems and Clean Energy*, pages 1128–1139, 2020b. doi: 10.35833/MPCE.2020.000522.

Yuhao Zhou, Bei Zhang, Chunlei Xu, Tu Lan, Ruisheng Diao, Di Shi, Zhiwei Wang, and Wei-Jen Lee. A data-driven method for fast ac optimal power flow solutions via deep reinforcement learning. *Journal of Modern Power Systems and Clean Energy*, 8(6): 1128–1139, 2020c. doi: 10.35833/MPCE.2020.000522.

Ray Daniel Zimmerman, Carlos E. Murillo-Sánchez, and Robert J. Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12–19, 2011. doi: 10.1109/TPWRS.2010.2051168.