National Technical University of Athens

School of Electrical and Computer Engineering

Division of Signals, Control and Robotics

Decision-Making in Stochastic Environments Using Diffusion Models

DIPLOMA THESIS of

Stelios Zarifis

Supervisor:
Petros Maragos, Professor
NTUA

Co-Supervisor:
Ioannis Kordonis, Assistant Professor
NTUA

COMPUTER VISION, SPEECH COMMUNICATION AND SIGNAL PROCESSING GROUP

Athens, June 2025

National Technical University of Athens

School of Electrical and Computer Engineering

Division of Signals, Control and Robotics

Decision-Making in Stochastic Environments Using Diffusion Models

DIPLOMA THESIS of

Stelios Zarifis

Supervisor:
Petros Maragos, Professor
NTUA

Co-Supervisor:
Ioannis Kordonis, Assistant Professor
NTUA

Approved by the Examination Committee on June 18th, 2025.

Petros Maragos	Ioannis Kordonis	Gerasimos Potamianos
Professor, NTUA	Assistant Professor, NTUA	Associate Professor, UTH
.....	.....	.....

Athens, June 2025

.....

Stelios Zarifis
Graduate of Electrical and Computer Engineering NTUA

Περίληψη

Η διαδοχική λήψη αποφάσεων υπό αβεβαιότητα αποτελεί απαιτητική πρόκληση σε πολλές εφαρμογές, και οι κλασικές μέθοδοι βελτιστοποίησης συχνά δεν μπορούν να διαχειριστούν την πολυπλοκότητα της δυναμικής στοχαστικών συστημάτων, οδηγώντας σε υποβέλτιστο έλεγχο. Στην παρούσα Διπλωματική Εργασία μελετάται η ενσωμάτωση πιθανοτικών προβλέψεων με βάση τα μοντέλα διάχυσης στον Προβλεπτικό Έλεγχο για τη βελτίωση της λήψης αποφάσεων σε μερικώς παρατηρήσιμα, στοχαστικά συστήματα.

Συγκεκριμένα, αναπτύσσουμε το πλαίσιο *Diffusion–Informed Model Predictive Control* (*D–I MPC*), το οποίο ενσωματώνει τις ισχυρές πιθανοτικές προβλέψεις των μοντέλων διάχυσης στον Προβλεπτικό Έλεγχο. Η προσέγγισή μας παράγει ένα σύνολο προσομοιώσεων της εξέλιξης του συστήματος προς έλεγχο, μέσω ενός μοντέλου διάχυσης και στη συνέχεια εφαρμόζει διάφορες παραλλαγές Προβλεπτικού Ελέγχου: ντετερμινιστικός έλεγχος, στοχαστικός έλεγχος, έλεγχος με δέντρα σεναρίων σε πολλαπλά στάδια και έλεγχος εμπλουτισμένος με ευρετικές.

Δείχνουμε την αποτελεσματικότητα του *D–I MPC* σε ένα σενάριο ενεργειακής εξισορροπητικής κερδοσκοπίας με χρήση ενός συστήματος αποθήκευσης ενέργειας στην αγορά ηλεκτρικής ενέργειας της Νέας Υόρκης. Το *D–I MPC* υπερέχει συστηματικά των υλοποιήσεων Προβλεπτικού Ελέγχου που στηρίζονται σε κλασικά προβλεπτικά μοντέλα και έναντι μεθόδων ενισχυτικής μάθησης χωρίς μοντέλο, και επιπλέον προσεγγίζει την απόδοση ιδεατών υλοποιήσεων που υποθέτουν τέλειες προβλέψεις στη διαδικασία βελτιστοποίησης.

Λέξεις-κλειδιά: Μοντέλα Διάχυσης, Πρόβλεψη Χρονοσειρών, Προβλεπτικός Έλεγχος, Ποσοτικοποίηση Αβεβαιότητας, Δέντρα Σεναρίων, Μηχανική Μάθηση, Βαθιά Μάθηση, Ενισχυτική Μάθηση, Αγορές Ενέργειας.

Abstract

Sequential decision-making under uncertainty is a difficult task in many real-world applications, and standard optimization methods often fail to capture complex stochastic dynamics, leading to suboptimal control. This thesis investigates the integration of diffusion-based probabilistic forecasting in Model Predictive Control (MPC) to enhance decision-making in partially observable, stochastic systems.

In this Thesis, we develop *Diffusion-Informed Model Predictive Control* (*D–I MPC*), a unified framework that integrates powerful diffusion-based probabilistic forecasting into MPC. Our approach generates an ensemble of future trajectories for the evolution of the system, using a diffusion model and then applies several MPC variants: deterministic MPC, stochastic MPC, multi-stage scenario tree-based MPC, and heuristic-augmented MPC.

We demonstrate the effectiveness of *D–I MPC* on an energy-arbitrage task with a battery energy storage system in the New York day-ahead electricity market, where it consistently outperforms MPC implementations driven by classical forecasters and model-free reinforcement-learning baselines, and additionally it performs closely to idealized implementations that use perfect forecasts in their optimization processes.

Keywords: Diffusion Models, Time Series Forecasting, Model Predictive Control, Uncertainty Quantification, Scenario Trees, Machine Learning, Deep Learning, Reinforcement Learning, Energy Markets.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον Καθηγητή κ. Πέτρο Μαραγκό, ο οποίος μου έδωσε την ευκαιρία να ασχοληθώ με ένα τόσο ενδιαφέρον και σύγχρονο θέμα και με υποστήριξε καθ' όλη τη διάρκεια της έρευνάς μου. Ευχαριστώ επίσης τον Επίκουρο Καθηγητή κ. Ιωάννη Κορδώνη για τη συνεπή καθοδήγησή του και τις πολύτιμες συμβουλές που μου προσέφερε, τόσο για την έρευνα της παρούσας εργασίας, όσο και για το μέλλον μου.
Θερμές ευχαριστίες οφείλω και στην οικογένειά μου: στους γονείς μου και τον αδερφό μου για την αμέριστη στήριξη και την πίστη στις δυνατότητές μου, και στους παππούδες μου για την αγάπη και τη φροντίδα που μου προσέφεραν όλα αυτά τα χρόνια. Η συμβολή τους ήταν καθοριστική για την επιτυχή ολοκλήρωση των σπουδών μου.

Στέλιος Ζαρίφης
Ιούνιος 2025

Στην οικογένειά μου

Contents

List of Figures

List of Tables

Chapter 1

Εκτενής Περίληψη στα Ελληνικά

Contents

1.1 Εισαγωγή

1.1.1 Το Πρόβλημα της Λήψης Αποφάσεων υπό Αβεβαιότητα

Η δημιουργία νοήμονων μηχανών απασχολεί την ανθρωπότητα από τα αρχαία χρόνια. Η ελληνική μυθολογία περιέχει πληθώρα ιστοριών για τέτοια μηχανικά όντα, όπως ο μύθος του χάλκινου γίγαντα Τάλου που φρουρούσε την Κρήτη και οι αυτόματοι τρίποδες του Ηφαίστου που υπηρετούσαν τους θεούς. Στην ιστορία, φιλόσοφοι και εφευρέτες όπως ο Αρχύτας ο Ταραντίνος και ο Ήρων ο Αλεξανδρινός σχεδίασαν αυτόματα που μιμούνταν ζωντανά όντα ή εκτελούσαν σύνθετες λειτουργίες. Η επίσημη καθιέρωση της Τεχνητής Νοημοσύνης (ΤΝ) ως επιστημονικό πεδίο έγινε δύο χιλιετίες μετά, το 1956 στη Διάσκεψη του Ντάρτμουθ (Dartmouth), όπου καθιερώθηκε ο όρος "Τεχνητή Νοημοσύνη" ("Artificial Intelligence") από τον Τζον Μακάρθι. Στα πρώτα βήματά της, η ΤΝ περιορίστηκε σε συμβολικές μεθόδους, προσπαθώντας να μιμηθεί τη λογική του ανθρώπου ορίζοντας κανόνες.

Οι περιορισμοί αυτών των προσεγγίσεων σε πραγματικές εφαρμογές, λόγω της μεταβλητότητας και αβεβαιότητας του πραγματικού κόσμου, οδήγησαν στην ανάπτυξη της Μηχανικής Μάθησης (ΜΜ) και της Βαθιάς Μάθησης (ΒΜ). Αυτά τα συστήματα επιτρέπουν τη μάθηση από δεδομένα αντί να προγραμματίζονται ρητά. Πρώτες εφαρμογές της τεχνητής ήταν το πρόγραμμα που παίζει ντάμα του Άρθουρ Σάμουελ και το Perceptron του Φρανκ Ρόζενμπλαττ. Η ΜΜ ξεκίνησε να εξελίσσεται ραγδαία τις δεκαετίες των 1980 και 1990 και συνεχίζει με εκθετικό ρυθμό. Πιο πρόσφατα, το ενδιαφέρον εστιάζεται στα Μεγάλα Γλωσσικά Μοντέλα (LLMs) και τα Μοντέλα Διάχυσης (Diffusion Models) που έχουν φέρει επανάσταση στην παραγωγή φυσικής γλώσσας και δεδομένων υψηλής ποιότητας, αντίστοιχα.

Σήμερα, η ΤΝ, η ΜΜ και η ΒΜ βρίσκουν εφαρμογή σε πολλά καθημερινά συστήματα, από την αναγνώριση προσώπων και την επεξεργασία φυσικής γλώσσας, έως και την έξυπνη λήψη αποφάσεων σε πολύπλοκα στοχαστικά συστήματα, όπως τα δίκτυα ενέργειας και οι χρηματοπιστωτικές αγορές. Σε τέτοια περιβάλλοντα, η ΤΝ μπορεί να επεξεργάζεται τεράστιους όγκους δεδομένων και να αναγνωρίζει κρυφά μοτίβα, τα οποία επηρεάζονται από πολλαπλούς παράγοντες, που είναι αδύνατο να συλληφθούν από την ανθρώπινη διαίσθηση ή τα παραδοσιακά μοντέλα. Αυτό έχει οδηγήσει σε σημαντικές εφαρμογές των μοντέλων της ΤΝ στην πρόβλεψη χρονοσειρών, όπου τα μοντέλα εκπαιδεύονται για να προβλέπουν μελλοντικές τιμές βασισμένα σε ιστορικές παρατηρήσεις, καθώς και στη βελτιστοποίηση συστημάτων μέσω της Ενισχυτικής Μάθησης (Reinforcement Learning), όπου ευφυείς πράκτορες (agents) μαθαίνουν να λαμβάνουν αποτελεσματικές διαδοχικές αποφάσεις για την επίτευξη ενός στόχου.

Στα πραγματικά συστήματα, η δυναμική είναι σύνθετη και αβέβαιη και η εξέλιξή τους επηρεάζεται κυρίως από στοχαστικούς παράγοντες. Εφαρμογές όπως η διαχείριση ενέργειας, οι χρηματοπιστωτικές συναλλαγές και τα αυτόνομα οχήματα απαιτούν από τους πράκτορες να λαμβάνουν διαδοχικές αποφάσεις βάσει μερικών και θορυβωδών παρατηρήσεων, χωρίς την πλήρη γνώση της δυναμικής του συστήματος. Αυτές οι περιπτώσεις μοντελοποιούνται με Μερικώς Παρατηρήσιμα Περιβάλλοντα (Partially Observable Environments), όπου η λήψη αποφάσεων βασίζεται σε ελλιπείς πληροφορίες.

Τα παραδοσιακά μοντέλα πρόβλεψης χρονοσειρών, όπως τα αυτοπαλινδρομικά (autoregressive) μοντέλα, χρησιμοποιούνται ευρέως με αρκετή επιτυχία. Ωστόσο, η εγγενής τους απλότητα (συχνά υποθέτουν γραμμικότητα και στασιμότητα) περιορίζει την αποτελεσματικότητά τους στη σύλληψη πολύπλοκων μοτίβων στα δεδομένων του πραγματικού κόσμου. Μοντέλα Μηχανικής Μάθησης, όπως τα Επαναλαμβανόμενα Νευρωνικά Δίκτυα (RNNs) και τα Δίκτυα Μακράς Βραχυπρόθεσμης Μνήμης (LSTMs), αξιοποιούν την πολυπλοκότητα των νευρωνικών δικτύων για να μοντελοποιήσουν καλύτερα τα περίπλοκα δεδομένα. Πρόσφατα, τα μοντέλα διάχυσης επιδεικνύουν εντυπωσιακές

δυνατότητες στη μοντελοποίηση σύνθετων κατανομών δεδομένων, καθιστώντας τα κατάλληλα για την πιθανοτική πρόβλεψη χρονοσειρών, οδηγώντας σε ακόμη μεγαλύτερη ακρίβεια.

Στην Ενισχυτική Μάθηση, οι πράκτορες μαθαίνουν βέλτιστες πολιτικές μέσω αλληλεπιδράσεων με το περιβάλλον για την επίτευξη ενός στόχου. Σε μερικώς παρατηρήσιμα περιβάλλοντα, οι αποφάσεις πρέπει να βασίζονται σε ατελείς πληροφορίες. Οι προσεγγίσεις Ενισχυτικής Μάθησης Χωρίς Μοντέλο (Model-Free RL) μαθαίνουν απευθείας από την εμπειρία των αλληλεπιδράσεων, χωρίς τη μοντελοποίηση της δυναμικής του περιβάλλοντος. Το πλεονέκτημά τους είναι η απλότητα και η δυνατότητα ευρείας εφαρμογής, καθώς δεν απαιτούν πρότερη γνώση της δυναμικής ή τον υπολογιστικό φόρτο κατασκευής ενός μοντέλου του συστήματος. Ωστόσο, για να συγκλίνουν σε αποτελεσματικές συμπεριφορές, συνήθως απαιτούν τεράστιο αριθμό αλληλεπιδράσεων με το περιβάλλον. Ενώ οι αλγόριθμοι Ενισχυτικής Μάθησης Χωρίς Μοντέλο μπορούν να προσεγγίσουν τη βέλτιστη συμπεριφορά, και θεωρητικά να την επιτύχουν με επαρκείς επαναλήψεις εκπαίδευσης και πολυπλοκότητα, η Ενισχυτική Μάθηση Βασισμένη σε Μοντέλο (Model-Based RL) αποδίδει καλύτερα σε λίγα βήματα εκπαίδευσης, καθώς ενσωματώνει το βήμα εκμάθησης ενός μοντέλου της δυναμικής του περιβάλλοντος, το οποίο μπορεί να χρησιμοποιεί για να σχεδιάζει τις κινήσεις του στο μέλλον, οδηγώντας σε ταχύτερη σύγκλιση σε επιθυμητή συμπεριφορά. Ωστόσο, η ακριβής μοντελοποίηση της δυναμικής είναι δύσκολη, ειδικά σε περιβάλλοντα υψηλών διαστάσεων και μερικής παρατηρησιμότητας.

Η ενσωμάτωση ισχυρών πιθανοτικών μοντέλων πρόβλεψης σε συστήματα λήψης αποφάσεων μπορεί να βελτιώσει σημαντικά την απόδοση σε αβέβαια περιβάλλοντα. Χρησιμοποιώντας μοντέλα διάχυσης για την πρόβλεψη χρονοσειρών, οι πράκτορες μπορούν να δημιουργήσουν ρεαλιστικά σενάρια για την εξέλιξη του συστήματος, ώστε να σχεδιάσουν τις ενέργειές τους, παρουσία αβεβαιότητας. Η ενσωμάτωση αυτών των πιθανοτικών προβλέψεων σε αλγορίθμους ελέγχου, οδηγεί σε σθεναρές αποφάσεις που λαμβάνουν υπόψη το ρίσκο και αυξάνει την αποδοτικότητα της εκπαίδευσης εφόσον αξιοποιεί τις παραγόμενες τροχιές.

1.1.2 Σκοπός και Συνεισφορές της Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία διερευνά την ενσωμάτωση πιθανοτικών μοντέλων πρόβλεψης βασισμένων σε μοντέλα διάχυσης σε αλγορίθμους διαδοχικής λήψης αποφάσεων, με τελικό σκοπό την παροχή ενός σθεναρού και αποτελεσματικού μοντέλου για έναν πράκτορα σε μερικώς παρατηρήσιμα στοχαστικά περιβάλλοντα. Μέσω εκτεταμένων πειραμάτων, αποδεικνύουμε την αποδοτικότητα του προτεινόμενου πλαισίου. Επιπλέον, αξιολογούμε τις μεθόδους που προτείνουμε στο πρόβλημα της ενεργειακής εξισορροπητικής κερδοσκοπίας, για να υποστηρίξουμε την εφαρμοσιμότητα και αποτελεσματικότητά τους σε πολύπλοκα, αβέβαια συστήματα του πραγματικού κόσμου.

Οι κύριες συνεισφορές της εργασίας είναι:

1. Προσαρμογή μοντέλων διάχυσης για πρόβλεψη χρονοσειρών σε μερικώς παρατηρήσιμα στοχαστικά δυναμικά συστήματα, με σκοπό τη δημιουργία πιθανοτικών προβλέψεων.

2. Ανάπτυξη ντετερμινιστικών και στοχαστικών αλγορίθμων βελτιστοποίησης που χρησιμοποιούν δειγματοληπτικές τροχιές από προβλεπτικό μοντέλο που βασίζεται σε μοντέλα διάχυσης με σκοπό το βέλτιστο έλεγχο.

3. Πρόταση ενός αλγορίθμου βελτιστοποίησης βασισμένου σε δέντρα σεναρίων για τη βελτιστοποίηση των δράσεων του πράκτορα μέσω μιας δομής που οργανώνει ιεραρχικά την αβεβαιότητα, προσφέροντας τη δυνατότητα λήψης αποφάσεων με επίγνωση ρίσκου.

4. Πρόταση ενός αλγορίθμου που χρησιμοποιεί μια ευρετική για την έμμεση επέκταση του τέλους του ορίζοντα βελτιστοποίησης, για πιο μακροπρόθεσμο σχεδιασμό.

5. Επίδειξη της αποτελεσματικότητας των προτεινόμενων μεθόδων μέσω μιας ολοκληρωμένης μελέτης περίπτωσης (case study) στην ενεργειακή εξισορροπητική κερδοσκοπία, αναδεικνύοντας βελτιώσεις στη συμπεριφορά των πρακτόρων σε σχέση με κλασικές μεθόδους πρόβλεψης και Ενισχυτικής Μάθησης Χωρίς Μοντέλο.

6. Μέρος της έρευνάς μας έχει γίνει δεκτό για δημοσίευση στο συνέδριο EUSIPCO 2025 [9].

1.2 Θεωρητικό Υπόβαθρο

1.2.1 Βασικές Έννοιες Βαθιάς Μάθησης και Χρονοσειρών

Βαθιά Μάθηση

Η Βαθιά Μάθηση (ΒΜ) αποτελεί υποσύνολο της Μηχανικής Μάθησης (ΜΜ) και χρησιμοποιεί βαθιά νευρωνικά δίκτυα (DNNs) (νευρωνικά δίκτυα με πολλές στρώσεις νευρώνων) για τη μοντελοποίηση σύνθετων δεδομένων. Αυτά τα δίκτυα αποτελούνται από πολλαπλά συνδεδεμένα στρώματα νευρώνων, που επεξεργάζονται μεγάλους όγκους δεδομένων για να μάθουν σχέσεις στα δεδομένα αυτά. Τα βαθιά νευρωνικά δίκτυα αποτελούνται από αλληλουχίες γραμμικών μετασχηματισμών και μη γραμμικές συναρτήσεις ενεργοποίησης. Κατά την προώθηση προς τα εμπρός (forward propagation), κάθε στρώμα μετασχηματίζει την είσοδό του σε μια νέα αναπαράσταση, ενώ κατά την οπισθοδιάδωση (backpropagation), το δίκτυο υπολογίζει τις κλίσης της συνάρτησης κόστους ως προς κάθε βάρος και τα ενημερώνει, με σκοπό τη σταδιακή βελτίωση της απόδοσης του μοντέλου.

Το perceptron [10], ένα από τα πρώτα μοντέλα νευρωνικών δικτύων (Φρανκ Ρόζενμπλαττ, 1958), αποτελεί τη βάση για πιο σύνθετες αρχιτεκτονικές. Ένα νευρωνικό δίκτυο αποτελείται από διασυνδεμένα στρώματα νευρώνων, όπου κάθε νευρώνας εφαρμόζει μια συνάρτηση ενεργοποίησης στο άθροισμα των βεβαρημένων εισόδων του, εισάγοντας έτσι μη-γραμμικότητα. Τα πιο κοινά είναι τα δίκτυα πρόσθιας τροφοδότησης (feedforward networks), όπου τα δεδομένα ρέουν προς μία κατεύθυνση. Ο αλγόριθμος οπισθοδιάδοσης (backpropagation) είναι ο βασικός τρόπος εκπαίδευσης και λειτουργεί υπολογίζοντας το σφάλμα μεταξύ της πρόβλεψης και της πραγματικής τιμής και προσαρμόζοντας τα βάρη των συνδέσεων για να το ελαχιστοποιήσει. Ένα απλό δίκτυο πρόσθιας τροφοδότησης περιλαμβάνει ένα στρώμα εισόδου (το οποίο λαμβάνει δεδομένα), ένα πλήθος κρυφών στρωμάτων (που εξάγουν χαρακτηριστικά για τα δεδομένα) και ένα στρώμα εξόδου (παράγει τα αποτελέσματα, π.χ., ταξινομήσεις ή προβλέψεις). Η εκπαίδευση βασίζεται στον αλγόριθμο Κατάβασης Δυναμικού (π.χ., Στοχαστική Κατάβαση Δυναμικού - SGD) για την ελαχιστοποίηση μιας συνάρτησης απώλειας (π.χ., MSE, Cross-Entropy).

Τα Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks - CNNs) [11], ειδικά σχεδιασμένα για την αναγνώριση χωρικών σχέσεων σε δεδομένα (π.χ., εικόνες), χρησιμοποιούν φίλτρα (πυρήνες) για την εξαγωγή χαρακτηριστικών όπως ακμές και υφές. Βασικές λειτουργίες τους είναι η συνέλιξη (convolution), το padding (για διατήρηση διαστάσεων) και το pooling (για μείωση του χωρικού μεγέθους και αποφυγή υπερπροσαρμογής). Τα εξαγόμενα χαρακτηριστικά εισάγονται σε πλήρως συνδεδεμένα στρώματα νευρώνων για την τελική εργασία (π.χ., ταξινόμηση ή παλινδρόμηση).

Τα Επαναλαμβανόμενα Νευρωνικά Δίκτυα (Recurrent Neural Networks - RNNs) [12,13] έχουν σχεδιαστεί ειδικά για ακολουθιακά δεδομένα, όπως χρονοσειρές, όπου προηγούμενες είσοδοι (παρελθόν) επηρεάζουν τις μελλοντικές εξόδους. Ο πυρήνας της λειτουργίας των RNNs είναι η αναδρομι-

κή ενημέρωση της κρυφής τους κατάστασης (hidden state) σε κάθε χρονικό βήμα. Ωστόσο, τα RNNs αντιμετωπίζουν πρόβλημα με μακροπρόθεσμες εξαρτήσεις λόγω του προβλήματος της εξαφάνισης κλίσης (Vanishing Gradient problem) αλλά και της έκρηξης κλίσης (Exploding Gradient problem). Τα Δίκτυα Μακράς Βραχυπρόθεσμης Μνήμης (Long Short-Term Memory - LSTMs) [14] επεκτείνουν τα RNNs με τη χρήση μηχανισμών με πύλες (gating mechanisms) (πύλη εισόδου, πύλη λήθης, πύλη εξόδου) για να ρυθμίζουν τη ροή πληροφοριών, επιτρέποντας την αποτελεσματική "αποθήκευση" μακροπρόθεσμων εξαρτήσεων. Τα Gated Recurrent Units (GRUs) απλοποιούν την αρχιτεκτονική των LSTMs, συγχωνεύοντας ορισμένες πύλες και μειώνοντας την υπολογιστική πολυπλοκότητα. Τέλος, οι Transformers οι οποίοι χρησιμοποιούν μηχανισμούς προσοχής (attention mechanisms), επιτρέπουν την παράλληλη επεξεργασία ακολουθιών και επιτυγχάνουν εξαιρετική απόδοση στην πρόβλεψη χρονοσειρών.

Οι Autoencoders [15] είναι μοντέλα μη επιβλεπόμενης μάθησης που μαθαίνουν μια συμπιεσμένη αναπαράσταση (latent representation) των δεδομένων εισόδου, από την οποία μπορούν να τα ανακατασκευάσουν. Αποτελούνται από έναν κωδικοποιητή (encoder) που συμπιέζει τα δεδομένα και έναν αποκωδικοποιητή (decoder) που τα ανακατασκευάζει. Ο στόχος της εκπαίδευσης είναι η ελαχιστοποίηση της απώλειας ανακατασκευής (reconstruction loss), συνήθως χρησιμοποιώντας το Μέσο Τετραγωνικό Σφάλμα (MSE).

Οι Transformers [16] έφεραν επανάσταση στην μοντελοποίηση ακολουθιών χρησιμοποιώντας μηχανισμούς προσοχής (attention mechanisms). Μπορούν να επεξεργάζονται ολόκληρες ακολουθίες δεδομένων ταυτόχρονα, λύνοντας το πρόβλημα της λήθης (forgetting). Ο μηχανισμός αυτοπροσοχής (self-attention) υπολογίζει τις σχέσεις μεταξύ όλων των θέσεων στην ακολουθία εισόδου, επιτρέποντας στο μοντέλο να σταθμίζει τη σημαντικότητα των στοιχείων στην πρόβλεψη.

Χρονοσειρές

Χρονοσειρά είναι μια ακολουθία δεδομένων που μετρήθηκαν ή παρατηρήθηκαν σε διαδοχικά χρονικά σημεία. Βασικό χαρακτηριστικό τους είναι η διάταξη στο χρόνο, και κάθε παρατήρηση μπορεί να εξαρτάται από τις προηγούμενες. Άλλα χαρακτηριστικά είναι η Τάση (Trend), δηλαδή η μακροπρόθεσμη αύξηση ή μείωση των δεδομένων, η Εποχικότητα (Seasonality), που αναφέρεται σε ένα επαναλαμβανόμενο μοτίβο σε μια περίοδο, και η Στασιμότητα (Stationarity), όπου τα στατιστικά χαρακτηριστικά (π.χ., η μέση τιμή και η διακύμανση) παραμένουν σταθερά με την πάροδο του χρόνου. Οι χρονοσειρές μπορούν επίσης να είναι μονομεταβλητές (μία μόνο μεταβλητή ανά χρονικό σημείο) ή πολυμεταβλητές (πολλαπλές μεταβλητές).

Η προεπεξεργασία των δεδομένων είναι ένα κρίσιμο βήμα στην ανάλυση και πρόβλεψη χρονοσειρών. Περιλαμβάνει τον καθαρισμό δεδομένων (διαχείριση ελλιπών τιμών, ακραίων τιμών και θορύβου), την εφαρμογή τεχνικών κλιμάκωσης (scaling) όπως min-max ή standard scaling για τυποποίηση των δεδομένων, και το stationarization για την αφαίρεση τάσεων (π.χ., μέσω διαφοροποίησης - differentiation). Το feature engineering είναι ένα σύνολο τεχνικών που βελτιώνουν την απόδοση των μοντέλων δημιουργώντας πρόσθετες μεταβλητές, όπως lag features, rolling statistics και χρονικά χαρακτηριστικά (time-based features) (π.χ., ημέρα της εβδομάδας). Ακόμη, τα φασματικά χαρακτηριστικά (spectral features), όπως αυτά που προκύπτουν από τον Γρήγορο Μετασχηματισμό Fourier (FFT), δίνουν πληροφορία για τις συχνότητες και τα περιοδικά μοτίβα στις χρονοσειρές. Τέλος, η αποσύνθεση χρονοσειρών (decomposition) διαχωρίζει μια χρονοσειρά σε τρία βασικά συστατικά: την τάση (Trend), την εποχικότητα (Seasonality) και τα υπόλοιπα (Residuals). Τα τελευταία αντιπροσωπεύουν τις απρόβλεπτες και ανεξήγητες ή θορυβώδεις διακυμάνσεις μετά την αφαίρεση της τάσης και της εποχικότητας.

1.2.2 Παραγωγικά Μοντέλα για Χρονοσειρές

Τα Παραγωγικά Μοντέλα (Generative Models) είναι μια κατηγορία μοντέλων Μηχανικής Μάθησης που μαθαίνουν την κατανομή πιθανότητας των δεδομένων εκπαίδευσης, επιτρέποντάς τους να δημιουργούν νέα δεδομένα που μοιάζουν με τα αρχικά [17]. Ένα παραγωγικό μοντέλο μπορεί να περιγραφεί ως μια από κοινού κατανομή πιθανότητας $p(x)$ για δεδομένα x, ή ως υπό συνθήκη μοντέλο $p(x|c)$ αν εξαρτάται από πρόσθετες πληροφορίες (covariates c). Η δημιουργία δεδομένων προκύπτει με δειγματοληψία από την κατανομή που μαθαίνει το μοντέλο.

Αν και η Διπλωματική Εργασία επικεντρώνεται στα Μοντέλα Διάχυσης, είναι σημαντικό να αναφερθούν κάποια από τα θεμελιώδη παραγωγικά μοντέλα:

Μοντέλα Μίξης Γκαουσιανών (Gaussian Mixture Models - GMMs)

Τα GMMs [18] υποθέτουν ότι τα δεδομένα παράγονται από ένα μείγμα πολλών Γκαουσιανών κατανομών. Η πιθανότητα ενός σημείου x σε ένα GMM με K συνιστώσες δίνεται ως:

$$p(x|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k),$$

όπου π_k είναι τα βάρη του μείγματος, και $\mathcal{N}(x|\mu_k, \Sigma_k)$ είναι η Γκαουσιανή κατανομή της k-οστής συνιστώσας. Η εκπαίδευσή τους γίνεται με τον αλγόριθμο Expectation-Maximization (EM) [19].

Κρυφά Μαρκοβιανά Μοντέλα (Hidden Markov Models - HMMs)

Τα HMMs αναπαριστούν ακολουθιακά δεδομένα χρησιμοποιώντας κρυφές (μη παρατηρήσιμες) καταστάσεις. Ένα HMM ορίζεται από ένα σύνολο κρυφών καταστάσεων (hidden states), πιθανότητες μετάβασης (transition probabilities) μεταξύ καταστάσεων, πιθανότητες εκπομπής (emission probabilities) παρατηρήσεων από κάθε κρυφή κατάσταση, και πιθανότητες αρχικών καταστάσεων (initial probabilities).

Παραγωγικά Μοντέλα Βασισμένα σε RNN (RNN-based Generative Models)

Τα Επαναλαμβανόμενα Νευρωνικά Δίκτυα (RNNs) και οι παραλλαγές τους (LSTM, GRU) [12, 14, 20] εστιάζουν στην παραγωγή χρονοσειρών, καθώς μπορούν να συλλάβουν χρονικές εξαρτήσεις. Σε κάθε χρονικό βήμα t, η κρυφή κατάσταση h_t ενημερώνεται βάσει της προηγούμενης κατάστασης και της τρέχουσας εισόδου, και το επόμενο βήμα της ακολουθίας προκύπτει από το μοντέλο του RNN χρησιμοποιώντας την τρέχουσα κρυφή κατάσταση: $x_{t+1} \sim p(x_{t+1}|h_t)$.

Variational Autoencoders (VAEs)

Οι VAEs [21] είναι οι πιθανοτικές εκδοχές των Autoencoders. Υποθέτουν ότι τα δεδομένα x παράγονται από λανθάνουσες μεταβλητές (latent variables) z μέσω ενός αποκωδικοποιητή $p_\theta(x|z)$, όπου οι z δειγματοληπτούνται από μια απλή προγενέστερη κατανομή (prior distribution) $p(z)$ (συνήθως Γκαουσιανή). Ο στόχος τους είναι να προσεγγιστεί η μεταγενέστερη κατανομή (posterior distribution) $p_\theta(z|x)$ χρησιμοποιώντας μια variational κατανομή $q_\phi(z|x)$. Επειδή ο άμεσος υπολογισμός της πραγματικής μεταγενέστερης κατανομής $p_\theta(z|x)$ είναι ανέφικτος, οι VAEs μεγιστοποιούν ένα κατώτερο όριο της λογαριθμικής πιθανοφάνειας (log-likelihood) των δεδομένων, το Evidence Lower Bound (ELBO), το οποίο ορίζεται ως:

$$\log p_\theta(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)\|p(z))$$

Ο πρώτος όρος είναι η απώλεια ανακατασκευής (reconstruction loss), ενώ ο δεύτερος όρος D_{KL} είναι η απόκλιση Kullback-Leibler, η οποία κανονικοποιεί τον λανθάνοντα χώρο (latent space) ενθαρρύνοντας την $q_\phi(z|x)$ να παραμένει κοντά στην προγενέστερη κατανομή $p(z)$. Για την αποτελεσματική εκπαίδευση, χρησιμοποιείται το *reparameterization trick*: $z = \mu_\phi(x) + \sigma_\phi(x) \cdot \epsilon$, όπου $\epsilon \sim \mathcal{N}(0, I)$, επιτρέποντας τον υπολογισμό των κλίσεων (gradients) ως προς τις παραμέτρους της $q_\phi(z|x)$. Οι Denoising Autoencoders [22] εκπαιδεύονται να ανακατασκευάζουν την αρχική είσοδο από μια αλλοιωμένη (θορυβώδη) εκδοχή της, προσφέροντας ανθεκτικότητα στον θόρυβο, άρα και ευελιξία στην εξαγωγή χαρακτηριστικών σε θορυβώδη περιβάλλοντα.

Παραγωγικά Ανταγωνιστικά Δίκτυα (Generative Adversarial Networks - GANs)

Τα GANs [23] αποτελούνται από έναν Γεννήτορα (Generator) (G) και έναν Διαχωριστή (Discriminator) (D) που ανταγωνίζονται σε ένα minimax παίγνιο. Ο Γεννήτωρ παράγει τεχνητά δεδομένα $G(z)$ από τυχαίο θόρυβο z, προσπαθώντας να "ξεγελάσει" τον Διαχωριστή. Ο Διαχωριστής προσπαθεί να διακρίνει τα πραγματικά δεδομένα x από τα παραχθέντα $G(z)$. Ο στόχος εκπαίδευσης είναι:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)} \left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)} \left[\log\left(1 - D\left(G\left(z\right)\right)\right)\right]$$

Παρόλο που τα GANs μπορούν να παράγουν δείγματα υψηλής ποιότητας, η εκπαίδευσή τους μπορεί να είναι ασταθής και να μη συγκλίνει πάντα.

Normalizing Flows

Τα μοντέλα Normalizing Flows [24] χρησιμοποιούν μια ακολουθία αντιστρέψιμων μετασχηματισμών ώστε να μετατρέψουν μια απλή κατανομή (π.χ., Γκαουσιανή) στην πολύπλοκη κατανομή που περιγράφει τα δεδομένα.

Επεκτάσεις Παραγωγικών Μοντέλων για Χρονοσειρές

Τόσο τα GANs όσο και οι VAEs έχουν επεκταθεί για την επεξεργασία χρονοσειρών. Παραδείγματα τέτοιων μοντέλων είναι τα TimeGAN [25] και TS-GAN [26], τα οποία ενσωματώνουν RNNs στον Γεννήτορα και τον Διαχωριστή για να διαχειριστούν την ακολουθιακή φύση των δεδομένων. Αντίστοιχα, οι Variational Recurrent Autoencoders (VRAEs) [27] ενσωματώνουν τα RNNs σε αρχιτεκτονικές Variational Autoencoders για τη μοντελοποίηση χρονοσειρών.

1.2.3 Μοντέλα Διάχυσης και Εφαρμογές σε Χρονοσειρές

Τα Μοντέλα Διάχυσης (Diffusion Models) είναι μια κατηγορία παραγωγικών μοντέλων λανθανουσών μεταβλητών (latent variables) που βρίσκονται στην αιχμή της τεχνολογίας λόγω της ικανότητάς τους να μοντελοποιούν σύνθετες κατανομές δεδομένων με μεγάλη ακρίβεια. Εμπνευσμένα από ιδέες της θερμοδυναμικής μη-ισορροπίας (non-equilibrium thermodynamics) και της στατιστικής φυσικής [28], ορίζουν μια Μαρκοβιανή αλυσίδα μικρών βημάτων διάχυσης (diffusion steps) για να προσθέτουν σταδιακά θόρυβο στα δεδομένα, και στη συνέχεια μαθαίνουν να αντιστρέφουν αυτή τη διαδικασία για την ανακατασκευή τους [29].

Αρχή Λειτουργίας των Μοντέλων Διάχυσης

Η λειτουργία τους βασίζεται σε δύο κύριες διαδικασίες:

1. *Εμπρόσθια Διαδικασία Διάχυσης (Forward Diffusion Process)*: Η διαδικασία αυτή προσθέτει σταδιακά Γκαουσιανό θόρυβο στα δεδομένα σε διαδοχικά χρονικά βήματα $t = 1, \ldots, T$, καταστρέφοντας τη δομή τους. Η αρχική κατανομή δεδομένων $q(x_0)$, μετασχηματίζεται σταδιακά σε μια κατανομή που μοιάζει με Γκαουσιανή κατανομή στο βήμα T. Η μετάβαση σε κάθε βήμα είναι προσθέτει θόρυβο με βάση ένα χρονοδιάγραμμα διακύμανσης (variance schedule) β_t:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t\mathbf{I})$$

Η συνολική πιθανότητα για την ακολουθία $x_1, \ldots, x_T$ δεδομένου του x_0 δίνεται από:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

Μία σημαντική ιδιότητα είναι η δυνατότητα δειγματοληψίας του x_t σε οποιοδήποτε βήμα t από το x_0 σε κλειστή μορφή:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)\mathbf{I})$$

όπου $\alpha_t = 1 - \beta_t$ και $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$.

2. *Αντίστροφη Διαδικασία (Reverse Diffusion Process)*: Η διαδικασία αυτή παράγει τα δεδομένα. Ξεκινά από τυχαίο θόρυβο, ένα δείγμα από την προγενέστερη κατανομή (η οποία μοιάζει με την κατανομή των x_T), και μαθαίνει να αντιστρέφει τη Εμπρόσθια Διαδικασία Διάχυσης, αφαιρώντας σταδιακά τον θόρυβο για να ανακατασκευάζει τα αρχικά δεδομένα από τις θορυβώδεις εκδοχές τους. Η μετάβαση $p_\theta(x_{t-1} \mid x_t)$ μοντελοποιείται από ένα νευρωνικό δίκτυο με παραμέτρους θ:

$$p_\theta(x_{0:T}) := p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t),$$

όπου $p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$. Το νευρωνικό δίκτυο μαθαίνει τις παραμέτρους της μέσης τιμής μ_θ και της διακύμανσης Σ_θ για κάθε βήμα της διάχυσης. Ο στόχος είναι να μεγιστοποιηθεί η λογαριθμική πιθανοφάνεια των παραγόμενων δεδομένων, το οποίο γίνεται μέσω της ελαχιστοποίησης ενός άνω ορίου της αρνητικής λογαριθμικής πιθανοφάνειας (ELBO).

Μοντέλα Διάχυσης για Πρόβλεψη Χρονοσειρών

Ενώ τα μοντέλα διάχυσης χρησιμοποιήθηκαν αρχικά για εφαρμογές δημιουργίας και επεξεργασίας εικόνων, έχουν επεκταθεί και σε εργασίες χρονοσειρών, όπως η πρόβλεψη, η συμπλήρωση ελλιπών τιμών και η παραγωγή ακολουθιακών δεδομένων [30]. Εστιάζουμε στα μοντέλα τα οποία προβλέπουν πολυμεταβλητές χρονοσειρές, όπου ο στόχος είναι η πρόβλεψη μελλοντικών τιμών δεδομένων των παρελθουσών τιμών.

TimeGrad: Το TimeGrad [7] είναι ένα αυτοπαλινδρομικό μοντέλο που χρησιμοποιεί τα Denoising Diffusion Probabilistic Models (DDPMs) για την πιθανοτική πρόβλεψη πολυμεταβλητών

χρονοσειρών. Η βασική ιδέα είναι η μοντελοποίηση της υπό συνθήκη κατανομής των μελλοντικών χρονικών βημάτων δεδομένων παρελθόντων δεδομένων:

$$q_X(x_{t_0:T}|x_{1:t_0-1}, c_{1:T}) = \prod_{t=t_0}^{T} q_X(x_t|x_{1:t-1}, c_{1:T}),$$

όπου x_t είναι το πολυμεταβλητό διάνυσμα δεδομένων στο βήμα t και $c_{1:T}$ είναι γνωστές συμμεταβλητές (covariates), που μοντελοποιούν επιπλέον πληροφορία, όπως γεγονότα (events), η οποία επηρεάζει τη μορφή της χρονοσειράς. Κάθε παράγων στο γινόμενο μοντελοποιείται από ένα υπό συνθήκη μοντέλο διάχυσης (conditional diffusion model), με τις χρονικές εξαρτήσεις να κωδικοποιούνται μέσω μιας αρχιτεκτονικής RNN. Το TimeGrad υπερέχει έναντι άλλων μεθόδων σε διάφορα πραγματικά σύνολα δεδομένων, αφού χάρη στη χρήση διαδικασιών διάχυσης, έχει τη δυνατότητα να μοντελοποιεί πολύ σύνθετες χρονικές εξαρτήσεις.

ScoreGrad: Το ScoreGrad [8] είναι ακόμα ένα μοντέλο πρόβλεψης χρονοσειρών βασισμένο σε διαδικασίες διάχυσης, το οποίο θεωρεί τη διαδικασία διάχυσης σε συνεχή χρόνο χρησιμοποιώντας Στοχαστικές Διαφορικές Εξισώσεις (SDEs). Η αντίστροφη διαδικασία μοντελοποιείται από μια αντίστροφη SDE. Το ScoreGrad χρησιμοποιεί έναν δειγματολήπτη προβλέπτη-διορθωτή (predictor-corrector sampler) για την πρόβλεψη. Έχει επιτύχει εξαιρετικά αποτελέσματα, ελαφρώς καλύτερα από το TimeGrad, σε διάφορα σύνολα δεδομένων.

D3VAE (Diffusion, Denoise and Disentanglement BVAE): Το D3VAE [31] αντιμετωπίζει την πρόβλεψη θορυβωδών και ελλιπών χρονοσειρών χρησιμοποιώντας μια συζευγμένη διαδικασία διάχυσης για παράθυρα πλαισίου (context) και πρόβλεψης (prediction). Χρησιμοποιούν έναν αμφίδρομο Variational Autoencoder (Bidirectional VAE - BVAE) και ένα σύστημα (module) αποθορυβοποίησης (denoising score matching) για τη βελτίωση των προβλέψεων. Η συνάρτηση απώλειας του D3VAE συνδυάζει τέσσερις όρους: απώλεια αποκλίσεων KL (οι παραγόμενες θορυβώδεις εκδόσεις των χρονοσειρών που παράγει το μοντέλο ταιριάζουν στατιστικά με τις πραγματικές διαχυμένες χρονοσειρές από το σύνολο δεδομένων), απώλεια αποθορυβοποίησης (για την ανακατασκευή των δεδομένων), απώλεια ολικής συσχέτισης (total correlation loss - για να διαχωρίζονται οι λανθάνουσες μεταβλητές ώστε να αναπαριστούν ανεξάρτητα χαρακτηριστικά των χρονοσειρών) και μέσο τετραγωνικό σφάλμα. Ο στόχος είναι η μάθηση σθεναρών αναπαραστάσεων παρουσία αβεβαιότητας και η διάσπαση των λανθανουσών μεταβλητών για καλύτερη ερμηνευσιμότητα.

Άλλα μοντέλα που βασίζονται σε διάχυση για χρονοσειρές περιλαμβάνουν τα Diffusion Schrödinger Bridge (DSB) [32] για εκμάθηση πιθανοτικών μοντέλων, Diffusion Spatio-Temporal Graph (DiffSTG) [33] για χωροχρονικές χρονοσειρές με γραφήματα, και Graph Convolutional Recurrent Denoising Diffusion (GCRDD) [34] που συνδυάζει συνελικτικά δίκτυα γραφημάτων και επαναλαμβανόμενα δίκτυα για χωρικές και χρονικές εξαρτήσεις.

1.2.4 Ενισχυτική Μάθηση και Προβλεπτικός Έλεγχος

Ενισχυτική Μάθηση (Reinforcement Learning - RL)

Η Ενισχυτική Μάθηση (Reinforcement Learning - RL) [35] είναι μια προσέγγιση που επιτρέπει σε έναν πράκτορα (agent) μαθαίνει να λαμβάνει βέλτιστες αποφάσεις αλληλεπιδρώντας με το περιβάλλον (environment) του, εκτελώντας ενέργειες (actions) με βάση την τρέχουσα κατάσταση (state) του περιβάλλοντος και λαμβάνοντας ανατροφοδότηση (feedback) με τη μορφή ενός σήματος ανταμοιβής

(reward signal). Στόχος του πράκτορα είναι να μάθει μια πολιτική (policy), δηλαδή μια αντιστοίχιση από καταστάσεις σε ενέργειες, η οποία μεγιστοποιεί τη σωρευτική ανταμοιβή (cumulative reward). Η RL διαφέρει από την επιβλεπόμενη και μη-επιβλεπόμενη μάθηση, καθώς η εκμάθηση προκύπτει από τις συνέπειες των δράσεων του μέσω δοκιμών.

Μια μαθηματική διατύπωση του προβλήματος της RL γίνεται μέσα από τις Διαδικασίες Αποφάσεων Markov (Markov Decision Processes - MDPs). Βασικοί αλγόριθμοι για την προσέγγιση των λύσεων τέτοιων προβλημάτων είναι ο αλγόριθμος Temporal Difference learning (TD) [36] και το Q-learning [37]. Η ενσωμάτωση βαθιών νευρωνικών δικτύων οδήγησε στην Βαθιά Ενισχυτική Μάθηση (Deep Reinforcement Learning) , επιτρέποντας την αντιμετώπιση προβλημάτων με χώρους καταστάσεων και δράσεων που έχουν πολλές διαστάσεις.

1.2.5 Βασικές Έννοιες της RL

Διαδικασίες Αποφάσεων Markov (MDPs) Η Διαδικασία Απόφασης Markov (MDP) είναι το μαθηματικό πλαίσιο που περιγράφει τη δυναμική του περιβάλλοντος και την επίδραση του πράκτορα ως μια πεντάδα $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, όπου:

- $\mathcal{S}$: Σύνολο καταστάσεων.
- $\mathcal{A}$: Σύνολο δράσεων.
- $P(s'|s,a)$: Συνάρτηση πιθανότητας μετάβασης, όπου $P(s'|s,a)$ είναι η πιθανότητα μετάβασης στην κατάσταση s' από την s εκτελώντας τη δράση a.
- $R(s,a,s')$: Συνάρτηση ανταμοιβής που λαμβάνεται μετά τη μετάβαση από s σε s' με δράση a.
- $\gamma \in [0,1]$: Συντελεστής έκπτωσης, ο οποίος μειώνει την αξία των μελλοντικών ανταμοιβών.

Ο στόχος ενός πράκτορα RL είναι να βρει μια βέλτιστη πολιτική $\pi^*(s)$ που μεγιστοποιεί την αναμενόμενη συνολική ανταμοιβή:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

Συναρτήσεις Αξίας Οι συναρτήσεις αξίας ποσοτικοποιούν το "πόσο καλή" είναι μια κατάσταση ή ένα ζεύγος κατάστασης-δράσης όσον αφορά την αναμενόμενη σωρευτική ανταμοιβή.

- Η συνάρτηση αξίας κατάστασης ($V^\pi(s)$) δίνει την αναμενόμενη σωρευτική ανταμοιβή ξεκινώντας από την κατάσταση s και ακολουθώντας την πολιτική π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t | S_t = s]$$

- Η συνάρτηση αξίας κατάστασης-δράσης ($Q^\pi(s,a)$), συχνά αναφέρεται ως Q-συνάρτηση, ορίζει την αναμενόμενη αθροιστική ανταμοιβή από την εκτέλεση της δράσης a στην κατάσταση s και την επακόλουθη παρακολούθηση της πολιτικής π:

$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]$$

Η εξίσωση Bellman περιγράφει την σχέση μεταξύ των αναμενόμενων μελλοντικών ανταμοιβών ενός πράκτορα σε ένα περιβάλλον. Εκφράζει ότι η τρέχουσα αξία ενός ζεύγους κατάστασης-δράσης ($Q^*(s,a)$) εξαρτάται από την άμεση ανταμοιβή εφαρμογής της δράσης στην κατάσταση και τη μέγιστη αναμενόμενη αξία των επόμενων καταστάσεων. Η εξίσωση Bellman για τη βέλτιστη συνάρτηση Q^* (εφαρμόζοντας τη βέλτιστη πολιτική) είναι:

$$Q^*(s,a) = \mathbb{E}_{s' \sim P(s'|s,a)}[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')].$$

Δίλημμα Εξερεύνησης - Εκμετάλλευσης Ένα από τα βασικά προβλήματα στην Ενισχυτική Μάθηση είναι η ισορροπία μεταξύ εξερεύνησης (exploration) και εκμετάλλευσης (exploitation). Η εξερεύνηση αφορά τη δοκιμή νέων δράσεων, ενώ η εκμετάλλευση είναι η επιλογή δράσεων που ο πράκτορας έχει ήδη διαπιστώσει ότι είναι πιο ανταποδοτικές. Η βέλτιστη συνολική ανταμοιβή προσεγγίζεται ισορροπώντας τις δύο αυτές τακτικές.

Μερικώς Παρατηρήσιμες Διαδικασίες Αποφάσεων Markov (POMDPs) Σε πραγματικά σενάρια, ο πράκτορας έχει τη δυνατότητα να λαμβάνει μερικές ή θορυβώδεις παρατηρήσεις. Για αυτά τα προβλήματα, χρησιμοποιούνται οι Μερικώς Παρρατηρήσιμες Διαδικασίες Αποφάσεων Markov (Partially Observable Markov Decision Processes - POMDPs), όπου ο πράκτορας διατηρεί μια κατάσταση πεποίθησης (belief state) για την πραγματική, κρυφή κατάσταση.

Επίλυση Διαδικασιών Αποφάσεων Markov

Η επίλυση των MDPs αποσκοπεί στην εύρεση μιας βέλτιστης πολιτικής π^* που μεγιστοποιεί την αναμενόμενη σωρευτική ανταμοιβή.

Προσεγγίσεις για την Επίλυση Διακριτών MDPs Σε διακριτές MDPs, όπου οι χώροι καταστάσεων και δράσεων είναι πεπερασμένοι, είναι θεωρητικά δυνατό να αναπαρασταθούν οι συναρτήσεις αξίας σε μορφή πίνακα. Ωστόσο, σε πολλά προβλήματα, όπου ο χώρος καταστάσεων μπορεί να είναι τεράστιος, απαιτούνται προσεγγιστικές αναπαραστάσεις για την πιο αποδοτική επίλυσή τους.

Δυναμικός Προγραμματισμός (Dynamic Programming - DP): Οι μέθοδοι DP [38] χρησιμοποιούν τις εξισώσεις Bellman για να υπολογίσουν αναδρομικά βέλτιστες συναρτήσεις αξίας και πολιτικές.

- **Policy Iteration**: Η προσέγγιση αυτή εναλλάσσεται μεταξύ των εξής δύο φάσεων:
  1. Αξιολόγηση Πολιτικής (Policy Evaluation): Υπολογίζει τη συνάρτηση αξίας $V^\pi(s)$ για την τρέχουσα πολιτική π.
  2. Βελτίωση Πολιτικής (Policy Improvement): Ενημερώνει την πολιτική επιλέγοντας, για κάθε κατάσταση, τη δράση που μεγιστοποιεί την αναμενόμενη σωρευτική ανταμοιβή.

  Ο αλγόριθμος επαναλαμβάνεται μέχρι η πολιτική να συγκλίνει στη βέλτιστη πολιτική π^*.

- **Value Iteration**: Ενημερώνει επαναληπτικά τη συνάρτηση αξίας χρησιμοποιώντας την εξίσωση βελτιστότητας Bellman:

$$V(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$$

Όταν η συνάρτηση αξίας $V^*(s)$ συγκλίνει, η βέλτιστη πολιτική προκύπτει επιλέγοντας άπληστα (greedily) σε κάθε βήμα τη δράση που επιτυγχάνει τη μέγιστη αξία για κάθε κατάσταση.

Μέθοδοι Monte Carlo Οι μέθοδοι Monte Carlo χρησιμοποιούνται όταν δεν υπάρχει ένα διαθέσιμο μοντέλο του περιβάλλοντος. Εκτιμούν τη συνάρτηση αξίας χρησιμοποιώντας ως εκτιμήτρια τη μέση τιμή των παρατηρούμενων επιστροφών από δείγματα επεισοδίων. Υπάρχουν δύο παραλλαγές των μεθόδων Monte-Carlo:

- First-Visit Monte Carlo: Υπολογίζει τη μέση σωρευτική ανταμοιβή μόνο την πρώτη φορά που επισκέπτεται μια κατάσταση σε κάθε επεισόδιο. Είναι *αμερόληπτη* (unbiased) αλλά έχει *υψηλότερη διακύμανση* (higher variance) λόγω αξιοποίησης λιγότερων δειγμάτων.

- Every-Visit Monte Carlo: Υπολογίζει τη μέση σωρευτική ανταμοιβή κάθε φορά που επισκέπτεται μια κατάσταση. Έχει *χαμηλότερη διακύμανση* αλλά ενδέχεται να εισάγει *μικρή μεροληψία* (slight bias) εάν οι επαναλαμβανόμενες επισκέψεις εντός του ίδιου επεισοδίου δεν είναι ανεξάρτητες.

Ένα μειονέκτημα των μεθόδων Monte Carlo είναι ότι απαιτούν την ολοκλήρωση ενός επεισοδίου για να ενημερώσουν τις εκτιμήσεις της συνάρτησης αξίας, γεγονός που καθυστερεί τη μάθηση όταν τα επεισόδια είναι μεγάλα και την καθιστά αδύνατη όταν η εκπαίδευση συμβαίνει σε ένα μοναδικό επεισόδιο.

Μέθοδοι Temporal Difference Learning (TD) Οι μέθοδοι TD συνδυάζουν τα πλεονεκτήματα του δυναμικού προγραμματισμού και των μεθόδων Monte Carlo. Ενημερώνουν τις εκτιμήσεις με βάση την τρέχουσα ανταμοιβή και την εκτιμώμενη αξία της επόμενης κατάστασης.

- TD(0): Είναι η απλούστερη μορφή, όπου η συνάρτηση αξίας κατάστασης ενημερώνεται σε κάθε χρονικό βήμα σύμφωνα με τον κανόνα:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_t + \gamma V(s_{t+1}) - V(s_t)],$$

όπου το $r_t + \gamma V(s_{t+1}) - V(s_t)$ ονομάζεται σφάλμα TD και α είναι ο ρυθμός μάθησης. Οι ενημερώσεις TD γίνονται σε κάθε βήμα, επιτρέποντας την εκμάθηση online και από ελλιπή επεισόδια.

- SARSA (State-Action-Reward-State-Action): Είναι μια μέθοδος TD on-policy[1] η οποία για τη μάθηση συναρτήσεων ($Q(s, a)$) χρησιμοποιεί την επόμενη δράση (a_{t+1}) που πράγματι επιλέχθηκε υπό την τρέχουσα πολιτική:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- Q-learning: Είναι μια μέθοδος TD off-policy[2] για τη μάθηση βέλτιστων συναρτήσεων ($Q(s, a)$) που χρησιμοποιεί τη μέγιστη δυνατή αξία στην επόμενη κατάσταση, ανεξάρτητα από τη δράση που επιλέχθηκε:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

- TD(λ): Επεκτείνει τις μεθόδους TD χρησιμοποιώντας τα eligibility traces, τα οποία συσσωρεύουν credit[3] για προηγούμενες καταστάσεις ή ζεύγη καταστάσεων-δράσεων.

[1] Η πολιτική που χρησιμοποιείται για την επιλογή δράσεων είναι η ίδια με αυτήν που βελτιστοποιείται κατά την εκμάθηση.

[2] Η πολιτική που χρησιμοποιείται για τη λήψη των αποφάσεων (που συλλέγει δεδομένα) μπορεί να διαφέρει από αυτήν που βελτιστοποιείται.

[3] Ο όρος credit αναφέρεται στην "αναγνώριση" ή την απόδοση "ευθύνης" σε καταστάσεις ή ενέργειες που συνέβαλαν σε μια μελλοντική επιβράβευση.

Προσεγγίσεις για την Επίλυση Συνεχών MDPs Στον πραγματικό κόσμο, οι χώροι καταστάσεων και δράσεων μπορεί να είναι συνεχείς, με τις καταστάσεις και ενέργειες να αναπαρίστανται ως συνεχή διανύσματα $s_t \in \mathbb{R}^{D_s}$ και $a_t \in \mathbb{R}^{D_a}$. Αυτό δημιουργεί προβλήματα, καθώς οι λύσεις σε μορφή πίνακα δεν είναι εφικτές. Έτσι, οι προσεγγιστές συναρτήσεων (function approximators), όπως νευρωνικά δίκτυα, είναι απαραίτητοι για τη γενίκευση της πολιτικής ή της συνάρτησης αξίας από τις παρατηρηθείσες καταστάσεις και ενέργειες της εκπαίδευσης σε παρόμοιες, άγνωστες καταστάσεις και ενέργειες. Η μετάβαση σε νέες καταστάσεις μοντελοποιείται συχνά ως

$$s_{t+1} = T(s_t, a_t) + \omega_T(s_t, a_t),$$

όπου $T(s_t, a_t)$ είναι μια ντετερμινιστική συνάρτηση μετάβασης και $\omega_T(s_t, a_t)$ ένας όρος θορύβου. Σε συνεχή χώρο, οι εξισώσεις Bellman εκφράζονται με ολοκληρώματα αντί για αθροίσματα. Για παράδειγμα η εξίσωση βέλτιστότητας Bellman (Bellman optimality equation) ορίζεται ως:

$$V^*(s) \;=\; \max_{a \in \mathcal{A}} \int_{\mathcal{S}} P(s, a, s') \big[R(s, a, s') + \gamma \, V^*(s') \big] \, ds'.$$

Μέθοδοι Βελτιστοποίησης Πολιτικής (Policy Optimization Methods) οι μέθοδοι αυτές παραμετροποιούν απευθείας την πολιτική $\pi_\theta(a|s)$ και προσαρμόζουν τις παραμέτρους θ για να μεγιστοποιήσουν την αναμενόμενη σωρρευτική ανταμοιβή. Το θεώρημα κλίσης πολιτικής (policy gradient theorem) δίνει έναν τρόπο υπολογισμού της κλίσης της αναμενόμενης ανταμοιβής:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d^\pi, a \sim \pi_\theta}[\nabla_\theta Q^{\pi_\theta}(s, a)]$$

Παραδείγματα τέτοιων αλγορίθμων είναι ο Deep Deterministic Policy Gradient (DDPG) [39], ο οποίος επεκτείνει αυτή την ιδέα διατηρώντας μια ντετερμινιστική πολιτική (Actor) και έναν κριτικό (Critic) που μαθαίνει την συνάρτηση αξίας δράσης, και ο Proximal Policy Optimization (PPO) [40], που βελτιστοποιεί πιο ομαλά την πολιτική, χρησιμοποιώντας μια "clipped" αντικειμενική συνάρτηση (αποτρέποντας μεγάλες ενημερώσεις πολιτικής).

Μέθοδοι Actor-Critic Οι αλγόριθμοι Actor-Critic συνδυάζουν ιδέες από τις μεθόδους βασισμένες σε αξίες (value-based) και μεθόδους κλίσης-πολιτικής (policy-gradient). Ένα δίκτυο "κριτικού" (critic network) (π.χ., ένα δίκτυο αξίας Q_θ ή V_θ) αξιολογεί την απόδοση της πολιτικής, ενώ ένα δίκτυο "δράστη" (actor network) (π.χ., ένα δίκτυο πολιτικής π_ϕ) ενημερώνεται προς την κατεύθυνση που υποδεικνύεται από τον κριτικό. Οι εκτιμήσεις του κριτικού βοηθούν στη μείωση της διακύμανσης (variance) της κλίσης πολιτικής και η άμεση βελτιστοποίηση του actor οδηγεί σε πολιτικές με υψηλότερες επιστροφές. Ακόμα, τεχνικές όπως το experience replay[4] και τα target networks[5] χρησιμοποιούνται για τη βελτίωση της σταθερότητας της μάθησης.

Model-Free έναντι Model-Based Ενισχυτικής Μάθησης Οι προσεγγίσεις RL μπορούν να κατηγοριοποιηθούν και με βάση το αν ο πράκτορας έχει πρόσβαση σε ένα μοντέλο της δυναμικής του περιβάλλοντος:

- Ενισχυτική Μάθηση Βασισμένη σε Μοντέλο (Model-Based RL): Ο πράκτορας μαθαίνει ή διαθέτει ένα μοντέλο του συστήματος στο οποίο επιδρά (δηλαδή ένα μοντέλο που προσεγγίζει τις συναρτήσεις μετάβασης $P(s'|s, a)$ και ανταμοιβής $R(s, a)$). με αυτόν τον τρόπο μπορεί να προσομοιώνει μελλοντικές τροχιές των καταστάσεων του συστήματος και να σχεδιάζει τις

[4] Αποθήκευση και αναπαραγωγή προηγούμενων εμπειριών με στόχο τη μείωση της συσχέτισης μεταξύ των
[5] Χρήση καθυστερημένων (lagged) αντιγράφων των δικτύων για πιο ομαλή μάθηση

ενέργειές του. Το πλεονέκτημα αυτής της προσέγγισης είναι η επίτευξη καλής απόδοσης με δραματικά λιγότερες επαναλήψεις εκπαίδευσης. Ωστόσο, η δυσκολία του προβλήματος RL μετατοπίζεται από την εκμάθηση μιας καλής πολιτικής στην εκμάθηση ενός καλού μοντέλου.

- Ενισχυτική Μάθηση Χωρίς Μοντέλο (Model-Free RL): Αντίθετα, οι μέθοδοι Model-Free RL δε μαθαίνουν ρητά (explicitly) ένα μοντέλο για το περιβάλλον. Μαθαίνουν απευθείας τη συνάρτηση αξίας ($V(s)$ ή $Q(s,a)$) ή την πολιτική ($\pi(a|s)$) από τις αλληλεπιδράσεις με το περιβάλλον. Ωστόσο, συνήθως απαιτούν μεγάλο πλήθος αλληλεπιδράσεων με το περιβάλλον για να προσεγγίσουν βέλτιστες πολιτικές, ειδικά σε σύνθετα περιβάλλοντα, καθώς πρέπει να μάθουν από τις συνέπειες των δράσεων τους για να κατανοήσουν τη δυναμική.

Η επιλογή ανάμεσα σε Model-Free και Model-Based RL εξαρτάται από την πολυπλοκότητα του περιβάλλοντος, τη δυνατότητα απόκτησης ενός καλού μοντέλου του συστήματος και τον όγκο των δεδομένων διαθέσιμων για αλληλεπίδραση.

Προβλεπτικός Έλεγχος (Model Predictive Control - MPC)

Ο Προβλεπτικός Έλεγχος (Model Predictive Control - MPC) είναι μια μέθοδος ελέγχου συστημάτων που βασίζεται σε ένα μοντέλο της δυναμικής του συστήματος για να προβλέψει τη μελλοντική εξέλιξη σε έναν πεπερασμένο χρονικό ορίζοντα. Σε κάθε χρονικό βήμα, ο αλγόριθμος MPC επιλύει ένα πρόβλημα βέλτιστου ελέγχου ανοιχτού βρόχου[6] (open-loop optimal control), προσδιορίζοντας την ακολουθία των δράσεων ελέγχου (control actions) που βελτιστοποιεί μια αντικειμενική συνάρτηση (objective function), ενώ ταυτόχρονα ικανοποιεί τη δυναμική του συστήματος και τους περιορισμούς του.

Επιπλέον, το κεντρικό χαρακτηριστικό του MPC είναι η αρχή του κινούμενου ορίζοντα (receding horizon principle): από την υπολογιζόμενη βέλτιστη ακολουθία ελέγχου, μόνο η πρώτη δράση εφαρμόζεται στο σύστημα και στο επόμενο χρονικό βήμα, η διαδικασία βελτιστοποίησης επαναλαμβάνεται, με την κατάσταση στην οποία μετέβη το σύστημα να είναι η νέα αρχική συνθήκη για τον επόμενο ορίζοντα. Αυτή η επαναληπτική βελτιστοποίηση παρέχει σταθερότητα, καθώς ο ελεγκτής σχεδιάζει από την αρχή τις ενέργειές του όταν λαμβάνει νέες παρατηρήσεις, μειώνοντας έτσι τις διαταραχές.

Βασικά Στοιχεία του MPC Η γενική μορφή του MPC έχει τρία βασικά μέρη:

1. **Μοντελοποίηση Συστήματος:** Το MPC απαιτεί ένα ρητό μοντέλο του συστήματος για την πρόβλεψη της εξέλιξής του. Το MPC μπορεί να εφαρμοστεί τόσο σε συστήματα διακριτού χρόνου όσο και σε συστήματα συνεχούς χρόνου.

   - Για ένα σύστημα διακριτού χρόνου, η δυναμική εκφράζεται συνήθως με εξισώσεις διαφορών (difference equations):

   $$x(k+1) = f(x(k), u(k), d(k)),$$

   όπου $x(k)$ είναι το διάνυσμα κατάστασης, $u(k)$ είναι το διάνυσμα εισόδου-ελέγχου και $d(k)$ δηλώνει διαταραχές στο σύστημα.

[6] Στον βέλτιστο έλεγχο ανοιχτού βρόχου, η βέλτιστη ακολουθία δράσεων υπολογίζεται εκ των προτέρων βάσει μοντέλου του συστήματος και εφαρμόζεται χωρίς χρήση ανατροφοδότησης (feedback) για την προσαρμογή των μελλοντικών δράσεων.

- Για ένα σύστημα συνεχούς χρόνου, η δυναμική περιγράφεται από διαφορικές εξισώσεις (differential equations):

$$\dot{x}(t) = f(x(t), u(t), d(t)),$$

όπου $x(t)$ είναι το διάνυσμα κατάστασης, $u(t)$ το διάνυσμα ελέγχου και $d(t)$ οι διαταραχές.

2. **Πρόβλεψη:** Χρησιμοποιώντας το μοντέλο του συστήματος και την αρχική (τρέχουσα) κατάσταση $x(t)$, ο αλγόριθμος MPC προβλέπει τη μελλοντική τροχιά του συστήματος πάνω σε έναν πεπερασμένο ορίζοντα πρόβλεψης μήκους N (ή T σε συνεχή χρόνο), ώστε να υπολογίσει την ακολουθία μελλοντικών δράσεων ελέγχου.

3. **Βελτιστοποίηση με Κινούμενο Ορίζοντα:** Σε κάθε χρονικό βήμα t, ο MPC επιλύει ένα πρόβλημα βέλτιστου ελέγχου για να βρει την ακολουθία ελέγχου U που βελτιστοποιεί μια αντικειμενική συνάρτηση J πάνω στον ορίζοντα πρόβλεψης, υπό τη δυναμική του μοντέλου του συστήματος και τους περιορισμούς αυτού. Μόλις υπολογιστεί η βέλτιστη ακολουθία δράσεων $\{u^*(k|t)\}$, μόνο η πρώτη δράση εφαρμόζεται στο σύστημα. Στο επόμενο χρονικό βήμα $t + 1$, η νέα κατάσταση παρατηρείται, και ολόκληρο το πρόβλημα του επιλύεται ξανά στον νέο, μετατοπισμένο ορίζοντα πρόβλεψης.

Εφαρμογές του Προβλεπτικού Ελέγχου Το MPC χρησιμοποιείται σε πολλούς τομείς της μηχανικής, όπως για παράδειγμα σε χημικές εγκαταστάσεις, στη ρομποτική, στην οδήγηση οχημάτων και στα δίκτυα ενέργειας. Η ικανότητά του να προσαρμόζεται στις μεταβολές του ελεγκτέου συστήματος τον καθιστά μια πολύ δημοφιλή επιλογή σε πολλές εφαρμογές. Παραδείγματα εφαρμογών είναι η προσαρμογή και ο έλεγχος ροών, θερμοκρασιών και συνθέσεων σε χημικές διεργασίες, στον σχεδιασμό κίνησης σε ρομποτική και αυτόνομα οχήματα, και την εξισορρόπηση συστημάτων ηλεκτρικής ενέργειας, συμπεριλαμβανομένης της ενσωμάτωσης ανανεώσιμων πηγών ενέργειας οι οποίες εισάγουν ισχυρές διακυμάνσεις στο σύστημα.

1.3 Προτεινόμενη Μεθοδολογία

1.3.1 Ορισμός Προβλήματος και Πιθανοτική Πρόβλεψη

Κίνητρο

Σε πολλές εφαρμογές του πραγματικού κόσμου, διαδοχικές αποφάσεις πρέπει να λαμβάνονται υπό συνθήκες αβεβαιότητας, βασισμένες σε μερική ή θορυβώδη πληροφορία. Το μέλλον του συστήματος δεν είναι πλήρως γνωστό, αλλά προβλέψεις για αυτό μπορούν να γίνουν με κάποιο βαθμό βεβαιότητας. Η παρούσα εργασία ερευνά την ενσωμάτωση των μοντέλων διάχυσης (diffusion models) για πιθανοτική πρόβλεψη χρονοσειρών σε αλγορίθμους βελτιστοποίησης για τη διαδοχική λήψη αποφάσεων, όπως ο Προβλεπτικός Έλεγχος (Model Predictive Control - MPC). Στόχος είναι η βελτίωση της διαδικασίας λήψης αποφάσεων, ειδικά σε συστήματα με αβέβαια δυναμική και υψηλής διάστασης, χρησιμοποιώντας τις προβλέψεις των μοντέλων διάχυσης ως ένα στοχαστικό μοντέλο του συστήματος.

Ορισμός Προβλήματος

Στα περισσότερα προβλήματα λήψης αποφάσεων στον πραγματικό κόσμο, η πραγματική κατάσταση (true state) του περιβάλλοντος, δηλαδή όλες οι απαραίτητες πληροφορίες για τη λήψη βέλτιστων

αποφάσεων, δεν είναι μετρήσιμη. Συνεπώς, οι αλγόριθμοι λήψης αποφάσεων βασίζονται σε θορυβώδεις ή μερικές παρατηρήσεις. Το πρόβλημα RL θα πρέπει να μοντελοποιηθεί ως μια Μερικώς Παρατηρήσιμη Διαδικασία Αποφάσεων Markov (Partially Observable Markov Decision Process - POMDP). Μια POMDP ορίζεται από μια 7-άδα $(S, A, T, R, \Omega, O, \gamma)$, όπου:

- S: Ο χώρος των (κρυφών) καταστάσεων του περιβάλλοντος.
- A: Το σύνολο των διαθέσιμων δράσεων του πράκτορα.
- $T : S \times A \to \Delta(S)$: Η συνάρτηση μετάβασης κατάστασης, η οποία δίνει την πιθανότητα μετάβασης στην κατάσταση s' από την s με δράση a.
- $R : S \times A \to \mathbb{R}$: Η συνάρτηση ανταμοιβής, η οποία δίνει την άμεση ανταμοιβή $R(s, a)$ με βάση την (μη παρατηρούμενη) κατάσταση s και την επιλεγμένη δράση a.
- Ω: Το σύνολο όλων των πιθανών παρατηρήσεων.
- $O : S \times A \to \Delta(\Omega)$: Η συνάρτηση παρατήρησης, η οποία καθορίζει την πιθανότητα $O(o|s', a)$ λήψης της παρατήρησης o όταν το περιβάλλον μεταβαίνει στην κατάσταση s' με τη δράση a.
- $\gamma \in [0, 1)$: Ο συντελεστής έκπτωσης (discount factor).

Σε κάθε χρονική στιγμή t, το περιβάλλον βρίσκεται σε μια κρυφή κατάσταση s_t. Ο πράκτορας λαμβάνει μια παρατήρηση $o_t \in \Omega$ (που παρέχει μερική πληροφορία για την κατάσταση), επιλέγει μια δράση $a_t \in A$, και το περιβάλλον μεταβαίνει σε μια νέα κατάσταση s_{t+1}, ενώ δίνει στον πράκτορα μια νέα παρατήρηση o_{t+1} και άμεση ανταμοιβή r_t.

Η προτεινόμενη μεθοδολογία εστιάζει σε προβλήματα όπου το σύστημα δεν είναι πλήρως παρατηρήσιμο και η δυναμική του μπορεί να διακριθεί σε ντετερμινιστικό και στοχαστικό μέρος, όπου το στοχαστικό μέρος εξελίσσεται ανεξάρτητα από τις δράσεις του πράκτορα και το ντετερμινιστικό μέρος. Άρα η παρατήρηση o_t μπορεί να αναλυθεί σε δύο συνιστώσες:

$$o_t = \langle o_t^d, o_t^s \rangle,$$

όπου o_t^d είναι η ντετερμινιστική συνιστώσα (π.χ., κατάσταση φόρτισης μιας μπαταρίας που επηρεάζεται άμεσα από τις ενέργειες του πράκτορα) και o_t^s είναι η στοχαστική συνιστώσα (π.χ., τιμές ενέργειας που ακολουθούν πολύπλοκες, στοχαστικές δυναμικές).

Για τον σχεδιασμό των δράσεων του πράκτορα, χρησιμοποιείται ένα μοντέλο για να προβλέψει τη στοχαστική συνιστώσα της τροχιάς των μελλοντικών παρατηρήσεων. Αυτές οι τροχιές χρησιμοποιούνται για τον υπολογισμό της βέλτιστης ακολουθίας δράσεων για τη μεγιστοποίηση της αναμενόμενης σωρευτικής ανταμοιβής. Η μέθοδος που προτείνουμε χρησιμοποιεί μοντέλα πρόβλεψης χρονοσειρών βασισμένα σε διάχυση, όπως το TimeGrad, για την πρόβλεψη της εξέλιξης του στοχαστικού μέρους των παρατηρήσεων. Αυτό παρέχει στον πράκτορα ένα μοντέλο που λαμβάνει υπόψη την αβεβαιότητα και μπορεί να ενσωματωθεί σε αλγόριθμους ελέγχου όπως ο MPC.

Πιθανοτική Πρόβλεψη για POMDPs

Στην προσέγγισή μας, η πιθανοτική πρόβλεψη της στοχαστικής συνιστώσας των παρατηρήσεων (δηλαδή, του o_k^s) πραγματοποιείται από ένα μοντέλο πρόβλεψης χρονοσειρών βασισμένο σε διάχυση (diffusion-based time series forecasting model), συγκεκριμένα το TimeGrad. Το TimeGrad μαθαίνει την δεσμευμένη κατανομή πιθανότητας των μελλοντικών στοχαστικών παρατηρήσεων δεδομένων των ιστορικών δεδομένων. Δεδομένου ενός ιστορικού παρατηρήσεων $o_{1:k_0-1}^s$, το TimeGrad

προσεγγίζει την κατανομή των μελλοντικών τιμών σε έναν ορίζοντα πρόβλεψης $k = k_0, ..., N$ ως εξής:

$$\prod_{k=k_0}^{N} q(o_k^s | o_{1:k-1}^s) \approx \prod_{k=k_0}^{N} p_\theta(o_k^s | h_{k-1}),$$

όπου p_θ είναι η κατανομή πρόβλεψης που μαθαίνει το μοντέλο (θ είναι οι παράμετροι του μοντέλου), και h_{k-1} είναι η κρυφή κατάσταση ενός RNN που κωδικοποιεί σημαντικές πληροφορίες από προηγούμενες παρατηρήσεις.

Εντός του πλαισίου POMDP, οι πιθανοτικές προβλέψεις χρησιμοποιούνται για την εύρεση του βέλτιστου ελέγχου του συστήματος. Ενώ η ντετερμινιστική συνιστώσα της παρατήρησης διέπεται από γνωστή δυναμική, η στοχαστική συνιστώσα εκτιμάται χρησιμοποιώντας το TimeGrad. Ο συνδυασμός της ντετερμινιστικής συνιστώσας (o_k^d) με τη στοχαστική συνιστώσα (o_k^s) οδηγούν στην ολική παρατηρήση:

$$\hat{o}_k = \langle o_k^d, \hat{o}_k^s \rangle.$$

Αυτή η ακολουθία προβλεπόμενων παρατηρήσεων ενσωματώνεται στη συνέχεια σε αλγόριθμους ελέγχου, όπως ο MPC και οι στοχαστικές του παραλλαγές, για τον προσδιορισμό της βέλτιστης ακολουθίας δράσεων που μεγιστοποιεί την αναμενόμενη σωρευτική ανταμοιβή.

1.3.2 Προβλεπτικός Έλεγχος με Μοντέλα Διάχυσης (D-I MPC)

Ντετερμινιστικός MPC με Μοντέλα Διάχυσης (Deterministic MPC with Diffusion Models)

Το πρώτο βήμα είναι η ενσωμάτωση του μοντέλου πρόβλεψης βασισμένου σε διάχυση (TimeGrad) σε ένα ντετερμινιστικό αλγόριθμο MPC για τον υπολογισμό της βέλτιστης ακολουθίας ελέγχου στο μερικώς παρατηρήσιμο περιβάλλον.

Σε κάθε χρονικό βήμα απόφασης, δεδομένου ενός παραθύρου παρατηρηθεισών στοχαστικών συνιστωσών $o_0^s, o_1^s, \ldots, o_k^s$, το TimeGrad παράγει μια κατανομή M προβλέψεων $\{\hat{o}_{k+1}^{s,(i)}\}_{i=1}^M$ για το επόμενο χρονικό βήμα. Αυτά τα δείγματα συγκεντρώνονται χρησιμοποιώντας έναν στατιστικό τελεστή, όπως ο διάμεσος ή ο μέσος όρος, για να παραχθεί μία σημειακή πρόβλεψη:

$$\hat{o}_{k+1}^s = \mathcal{F}(o_0^s, \ldots, o_k^s) = \text{median}\{\hat{o}_{k+1}^{s,(i)}\}_{i=1}^M$$

Η συνολική παρατήρηση του επόμενου βήματος $\hat{o}_{k+1} = \langle o_{k+1}^d, \hat{o}_{k+1}^s \rangle$ κατασκευάζεται υπολογίζοντας και το ντετερμινιστικό μέρος o_{k+1}^d μετά την επιλογή μιας δράσης a_k.

Χρησιμοποιώντας αυτόν τον τελεστή πρόβλεψης με αυτοπαλινδρομικό τρόπο (autoregressively), παράγεται μια προβλεπόμενη τροχιά παρατηρήσεων για έναν μελλοντικό ορίζοντα N βημάτων. Το ντετερμινιστικό πρόβλημα MPC, όταν ξεκινάει από το χρονικό σημείο k_0 και βελτιστοποιεί για N βήματα μπροστά, ορίζεται ως εξής:

$$\begin{aligned}
\underset{a_{k_0},\ldots,a_{N+k_0-1}}{\text{maximize}} \quad & \sum_{k=k_0}^{N+k_0-1} R(\hat{o}_k, a_k) \\
\text{subject to} \quad & o_0, \ldots, o_{k_0-1} \text{ are observed}, \\
& \hat{o}_{k+1} = \mathcal{F}(o_0, \ldots, o_k), \quad \text{for } k = k_0 - 1, \ldots, N + k_0 - 2, \\
& a_k \in \mathcal{A}(s_k), \quad \text{for } k = k_0, \ldots, N + k_0 - 1.
\end{aligned}$$

Εδώ, $R(\hat{o}_k, a_k)$ συμβολίζει την ανταμοιβή που λαμβάνεται κατά την εφαρμογή της δράσης a_k με βάση την προβλεπόμενη παρατήρηση $\hat{o}_k$. Με τη συγκέντρωση (aggregation) πολλαπλών πιθανοτικών προβλέψεων σε μία (μέσω διάμεσης ή μέσης τιμής), η διαχείριση της αβεβαιότητας γίνεται αποκλειστικά στο επίπεδο της πρόβλεψης. Κατά συνέπεια, το πρόβλημα βελτιστοποίησης του MPC γίνεται ντετερμινιστικό, καθώς βασίζεται σε μία μόνο προβλεπόμενη τροχιά μελλοντικών παρατηρήσεων.

Στοχαστικός MPC με Μοντέλα Διάχυσης (Stochastic MPC with Diffusion Models)

Για να ληφθεί πλήρως υπόψη η αβεβαιότητα στην εξέλιξη της μη παρατηρούμενης στοχαστικής συνιστώσας κατά τη διάρκεια της βελτιστοποίησης, επεκτείνουμε τον κλασικό MPC ενσωματώνοντας πολλαπλές τροχιές πρόβλεψης. Σε αυτή την εκδοχή, η κατανομή των μελλοντικών στοχαστικών παρατηρήσεων ενσωματώνεται ρητά (explicitly) στο πρόβλημα βελτιστοποίησης. Προτείνουμε δύο παραλλαγές του Στοχαστικού MPC (SMPC): μία που χρησιμοποιεί προσομοιώσεις Monte Carlo και μία που χρησιμοποιεί προβλέψεις οργανωμένες σε μια δομή δέντρου.

SMPC με Προσομοιώσεις Monte Carlo Στον Monte Carlo SMPC, η αβεβαιότητα μοντελοποιείται με τη δειγματοληψία M ανεξάρτητων τροχιών από την κατανομή πρόβλεψης του TimeGrad. Για κάθε σενάριο $i \in \{1, \ldots, M\}$, η μελλοντική στοχαστική συνιστώσα παράγεται αναδρομικά για κάθε σημείο του ορίζοντα πρόβλεψης $k = k_0, \ldots, N + k_0 - 1$ ως:

$$o_{k+1}^{s,(i)} \sim p_\theta(o_{k+1}^s | h_k).$$

Για κάθε σενάριο i, η πλήρης προβλεπόμενη παρατήρηση στο χρονικό βήμα $k+1$ είναι:

$$\hat{o}_{k+1}^{(i)} = \langle o_{k+1}^d, o_{k+1}^{s,(i)} \rangle,$$

όπου o_{k+1}^d υπολογίζεται ντετερμινιστικά από τις γνωστές δυναμικές του συστήματος. Η σωρευτική ανταμοιβή κατά μήκος της i-οστής τροχιάς, με βάση τις εφαρμοζόμενες δράσεις, ορίζεται ως:

$$J^{(i)} = \sum_{k=k_0}^{N+k_0-1} R(\hat{o}_k^{(i)}, a_k), \quad \text{where } \hat{o}_k^{(i)} = \langle o_k^d, \hat{o}_k^{s,(i)} \rangle.$$

Το πρόβλημα βελτιστοποίησης SMPC αναζητά την ακολουθία δράσεων $\{a_{k_0}, \ldots, a_{N+k_0-1}\}$ που μεγιστοποιεί την αναμενόμενη σωρευτική ανταμοιβή η οποία προσεγγίζεται από τη μέση σωρευτική ανταμοιβή όλων των M σενάριων:

$$\begin{aligned} \underset{a_{k_0},\ldots,a_{N+k_0-1}}{\text{maximize}} \quad & \frac{1}{M} \sum_{i=1}^{M} J^{(i)} \\ \text{subject to} \quad & o_0, \ldots, o_{k_0-1} \text{ are observed}, \\ & o_{k+1}^{s,(i)} \sim p_\theta(o_{k+1}^s | h_k), \quad k = k_0, \ldots, N + k_0 - 1, \quad \forall i, \\ & a_k \in \mathcal{A}, \quad k = k_0, \ldots, N + k_0 - 1. \end{aligned}$$

Αυτή η εκδοχή ενσωματώνει την αβεβαιότητα σε ολόκληρο τον ορίζοντα πρόβλεψης, λαμβάνοντας υπόψη πολλαπλές πραγματοποιήσεις (realizations) τροχιών. Αυτό οδηγεί σε πιο σθεναρό έλεγχο, καθώς η στοχαστική δυναμική λαμβάνεται υπόψη κατά τη διαδικασία βελτιστοποίησης.

1.3.3 Δέντρα Σεναρίων με Μοντέλα Διάχυσης (DST)

Ένας ακόμα τρόπος να ενσωματώσουμε την αβεβαιότητα στις προβλέψεις επεκτείνοντας τον MPC είναι χρησιμοποιώντας δέντρα σεναρίων (scenario trees). Αυτά τα δέντρα αναπαριστούν τη μελλοντική αβεβαιότητα με διακριτά μονοπάτια σε δομή δέντρου, με κάθε κόμβο να δηλώνει μια πιθανή πραγματοποίηση της στοχαστικής διαδικασίας σε ένα στάδιο. Είναι σημαντικό να τηρούνται οι περιορισμοί non-anticipativity, ώστε οι αποφάσεις να βασίζονται μόνο σε πληροφορίες που είναι διαθέσιμες μέχρι το τρέχον στάδιο.

Μέθοδοι Κατασκευής Δέντρων Σεναρίων με Μοντέλα Διάχυσης

Α. Κατασκευή με Ομαδοποίηση Προς τα Εμπρός (Forward Clustering) Το δέντρο δημιουργείται στάδιο προς στάδιο. Σε κάθε κόμβο, το μοντέλο διάχυσης παράγει M προβλέψεις για το επόμενο βήμα. Αυτές οι προβλέψεις ομαδοποιούνται (π.χ., με τον αλγόριθμο K-means), και το κεντροειδές (centroid) κάθε ομάδας αποτελεί τον αντιπρόσωπό της. Η πιθανότητα πραγματοποίησης κάθε κόμβου καθορίζεται από το ποσοστό των δειγμάτων που ανήκουν σε αυτόν. Για να αποφευχθεί η εκθετική επέκταση του δέντρου, μπορεί να γίνεται περικοπή του δέντρου κατά τη διαδικασία δημιουργίας του (on-line), για παράδειγμα διατηρώντας τα L πιο πιθανά σενάρια σε κάθε στάδιο. Η μέθοδος εξασφαλίζει τους περιορισμούς non-anticipativity με φυσικό τρόπο κατά την κατασκευή, καθώς οι προβλέψεις για κάθε κόμβο-παιδί (child-node) βασίζονται μόνο στις παρατηρήσεις και τις πραγματοποιήσεις των κόμβων προγόνων. Η διαδικασία περιγράφεται στον Αλγόριθμο 1.

Β. Κατασκευή με Ιεραρχική Ομαδοποίηση Προς τα Πίσω (Backward-Hierarchical Clustering) Αρχικά, λαμβάνονται M πλήρεις τροχιές για όλο τον ορίζοντα. Το δέντρο κατασκευάζεται από τα φύλλα προς τη ρίζα, ομαδοποιώντας επανειλημμένα τις παραχθείσες τροχιές σε μειούμενες χρονικές αναλύσεις (temporal resolutions). Στο τελευταίο στάδιο, οι M τροχιές ομαδοποιούνται σε K_D ομάδες. Στα προηγούμενα στάδια, οι τροχιές των ομάδων συγχωνεύονται και επανα-ομαδοποιούνται, διασφαλίζοντας ότι τα σύνολα των τροχιών μόνο συνενώνονται, ποτέ δεν διαχωρίζονται, διατηρώντας τον περιορισμό non-anticipativity. Ένας βρόχος τύπου Lloyd βελτιστοποιεί την ομαδοποίηση επιλέγοντας το πλήθος των συστάδων αυτόματα, χωρίς την ανάγκη αυθαίρετων κατωφλιών (thresholds). Η μέθοδος περιγράφεται στον Αλγόριθμο 2.

Επίλυση του Προβλήματος Βελτιστοποίησης

Μετά την κατασκευή του δέντρου, επιλύεται το ακόλουθο πολυσταδιακό πρόβλημα Στοχαστικού MPC (SMPC):

$$\begin{aligned} \underset{\{a_t^n\}}{\text{maximize}} \quad & \sum_{t=0}^{T-1} \sum_{n \in \mathcal{N}_t} P_t^n \, R\big(o_t^n, a_t^n\big) \\ \text{s.t.} \quad & o_0 \text{ given}, \quad o_{t+1}^n = f\big(o_t^n, a_t^n, \xi_{t+1}^n\big), \\ & a_t^n = a_t^m \quad \forall\, n, m \text{ with same history up to } t, \\ & a_t^n \in \mathcal{A}. \end{aligned}$$

Εδώ, $\mathcal{N}_t$ είναι το σύνολο των κόμβων στο στάδιο t, με πιθανότητα P_t^n, παρατήρηση o_t^n και απόφαση a_t^n. Η μη-προσδοκία επιβάλλεται εξισώνοντας τις αποφάσεις a_t σε όλα τα σενάρια που μοιράζονται τον ίδιο γονικό κόμβο.

MPC Ενισχυμένος με Ευρετική (Heuristic-Augmented MPC)

Για να ξεπεραστεί ο εγγενής περιορισμός του πεπερασμένου ορίζοντα στον τυπικό MPC, προτείνουμε τον HA-MPC (Heuristic-Augmented MPC). Αυτή η προσέγγιση ενσωματώνει μια ευρετική στο πέρας του ορίζοντα που πληροφορεί για το αναμενόμενο σωρρευτικό κέρδος μετά τον ορίζοντα πρόβλεψης, επιτρέποντας πιο μακροπρόθεσμο σχεδιασμό.

Η ευρετική βασίζεται σε ένα προβλεπτικό μοντέλο (π.χ., LSTM), το οποίο εκπαιδεύεται να προβλέπει τη βέλτιστη τερματική παρατήρηση $o^{\text{opt}}_{N+k_0-1}$:

$$o^{\text{opt}}_{N+k_0-1} = \text{LSTM}_\theta\Big(\hat{o}^s_{k_0:N+k_0-1},\, \hat{o}^s_{N+k_0:N+k_0+L-1}\Big)$$

Η προσέγγιση είναι επιθυμητή και σε εφαρμογές που απαιτούν τη λήψη γρήγορων αποφάσεων, καθώς η εκτίμηση της βέλτιστης τερματικής κατάστασης μπορεί να επιτρέψει μικρότερο ορίζοντα βελτιστοποίησης.

Το πρόβλημα βελτιστοποίησης του HA-MPC διατυπώνεται ως:

$$\begin{aligned}
\underset{a_{k_0},\ldots,a_{N+k_0-1}}{\text{maximize}} \quad & \sum_{k=k_0}^{N+k_0-1} R\big(o^s_k, a_k\big) - \gamma \left\| \hat{o}^s_{N+k_0-1} - o^{\text{opt}}_{N+k_0-1} \right\|^2, \\
\text{subject to} \quad & o_0, o_1, \ldots, o_{k_0-1} \text{ observed}, \\
& \hat{o}^s_{k+1} = \mathcal{F}(o^s_0, \ldots, o^s_k), \quad k = k_0, \ldots, N+k_0-1, \\
& a_k \in \mathcal{A}, \quad k = k_0, \ldots, N+k_0-1.
\end{aligned}$$

Ο συντελεστής γ καθορίζει αν ο περιορισμός τερματικής κατάστασης είναι αυστηρός (hard constraint) ($\gamma = \infty$) ή όχι (soft constraint) ($\gamma < \infty$).

Με την ενσωμάτωση της ευρετικής, το H-A MPC επεκτείνει έμμεσα (implicitly) τον ορίζοντα πρόβλεψης, επιτρέποντας αποφάσεις που λαμβάνουν υπόψη τόσο τα βραχυπρόθεσμα όσο και τα μακροπρόθεσμα αποτελέσματα.

Ενισχυτική Μάθηση Χωρίς Μοντέλο (Model-Free Reinforcement Learning)

Για την αξιολόγηση των προτεινόμενων μεθόδων, υλοποιήσαμε και τεχνικές RL χωρίς μοντέλο. Εξετάσαμε τρεις παραλλαγές πρακτόρων σε POMDPs, που διαφέρουν στην αναπαράσταση της κατάστασης του περιβάλλοντος:

- Ιδεατή Κατάσταση: Ο πράκτορας έχει (μη ρεαλιστικά) πρόσβαση στις πραγματικές μελλοντικές τιμές ενός ορίζοντα: $s_t = \Big[o_t,\ o^s_{t+1:t+K}\Big]$.
- Κατάσταση LSTM: Ένα δίκτυο LSTM χρησιμοποιεί ένα πλήθος από παρατηρήσεις $o_{t-H+1:t}$ για να μάθει μια εσωτερική αναπαράσταση $h_t = \text{LSTM}_\psi\big(o_{t-H+1:t},\, h_{t-1}\big)$. Η κατάσταση του πράκτορα είναι η παράθεση της τρέχουσας παρατήρησης με την κρυφή κατάσταση του LSTM $s_t = \big[o_t,\ h_t\big]$.
- Κατάσταση TimeGrad: η κρυφή κατάσταση του RNN του TimeGrad h^{TG}_t δίνει την κατάσταση του περιβάλλοντος: $s_t = \big[o_t,\ h^{\text{TG}}_t\big]$, όπου:

$$h^{\text{TG}}_t = \text{TimeGradRNN}_\theta\big(o^s_{t-H+1:t},\, h_{t-1}\big)$$

1.4 Πειραματικά Αποτελέσματα

1.4.1 Δεδομένα και Εφαρμογή

Η προτεινόμενη μεθοδολογία, Diffusion-Informed MPC, αξιολογείται στο σενάριο της ενεργειακής εξισορροπητικής κερδοσκοπίας (energy arbitrage) στην ημερήσια αγορά ηλεκτρικής ενέργειας της Νέας Υόρκης που ελέγχεται από τον οργανισμό NYISO (New York Independent System Operator). Σε αυτό το πλαίσιο, ο στόχος είναι ο έλεγχος ένος Συστήματος Αποθήκευσης Ενέργειας με Μπαταρίες (Battery Energy Storage System - BESS) [41] για να μεγιστοποιηθεί το μακροπρόθεσμο κέρδος, λαμβάνοντας διαδοχικές αποφάσεις αγοραπωλησίας υπό την αβεβαιότητα των μελλοντικών τιμών ενέργειας.

Περιγραφή Δεδομένων

Το dataset που χρησιμοποιήσαμε διατίθεται δημόσια από τον NYISO και περιέχει ιστορικά δεδομένα τιμών ηλεκτρικής ενέργειας (Locational Based Marginal Prices - LBMP) ανά ώρα και ζώνη, καλύπτοντας έξι χρόνια. Οι αναλύσεις των δεδομένων που παραθέτουμε αναλυτικά στο Κεφάλαιο 5.2 έδειξαν:

- Χρονικές Εξαρτήσεις και Μοτίβα: Οι χρονοσειρές παρουσιάζουν περιοδικότητα (κυρίως ημερήσιους κύκλους), μεταβλητή τάση και εποχική εξάρτηση (seasonal dependencies).
- Στατιστικά Στοιχεία: Υπάρχουν σημαντικές διακυμάνσεις τιμών, άρα και ευκαιρίες κέρδους.
- Κατανομή Τιμών και Έκτοπες Τιμές (Outliers): Οι κατανομές των τιμών εμφανίζουν θετική λογότητα (skewness), όπου οι διάμεσοι είναι χαμηλότεροι από τους μέσους όρους. Οι ακραίες τιμές (outliers) είναι μόνο υψηλές, συνεπώς ακριβή μοντέλα πρόβλεψης είναι απαραίτητα για την εκμετάλλευσή των.
- Συσχετίσεις: Υπάρχουν ισχυρές θετικές συσχετίσεις μεταξύ των τιμών και των ενεργειακών φορτίων στις περισσότερες περιοχές, υποδηλώνοντας την ανάγκη για πολυμεταβλητά μοντέλα πρόβλεψης.
- Αυτοσυσχέτιση (Autocorrelation) και Μερική Αυτοσυσχέτιση (Partial Autocorrelation): Οι γραφικές παραστάσεις έδειξαν ισχυρή ημερήσια περιοδικότητα (κορυφές ανά 24 βήματα), καθώς και βραχυπρόθεσμες αλλά και μακροπρόθεσμες εξαρτήσεις που δείχνουν πως τα δεδομένα έχουν πολυπλοκότητα στη δομή τους.
- Έλεγχος ADF (Augmented Dickey-Fuller Test): Τα αποτελέσματα φανέρωσαν ότι οι χρονοσειρές τιμών είναι στατικές (stationary) σε μικρά παράθυρα, κάτι που είναι γενικά ευνοϊκό για τα μοντέλα.
- Κυλιόμενες Στατιστικές (Rolling Statistics): Η ανάλυση έδειξε ομαλές μεταβολές στον κυλιόμενο μέσο όρο και την τυπική απόκλιση, ενισχύοντας το συμπέρασμα ότι οι χρονοσειρές είναι στάσιμες σε μικρά παράθυρα, αλλά και την παρουσία ανωμαλιών που σημειώνουν την ανάγκη για πολύπλοκα μοντέλα.
- Ιστογράμματα: Επιβεβαίωσαν το θετικό skewness των κατανομών των τιμών και έδειξαν ότι οι κατανομές στις περισσότερες περιοχές είναι μονοτροπικές (unimodal).

Αξιολόγηση Προβλέψεων του TimeGrad

Το μοντέλο TimeGrad [7] χρησιμοποιήθηκε για την πρόβλεψη 27 συσχετισμένων χρονοσειρών (τιμές ενέργειας και φορτίων). Μετά από προσεκτική ρύθμιση παραμέτρων, το TimeGrad έδειξε εξαιρετική ακρίβεια πρόβλεψης για μελλοντικές ωριαίες τιμές έως και για 10 ημέρες στο μέλλον (240 βήματα). Ένα παράδειγμα των πιθανοτικών προβλέψεων φαίνεται στην Εικόνα 1.1. Ακόμα, υπερέχει

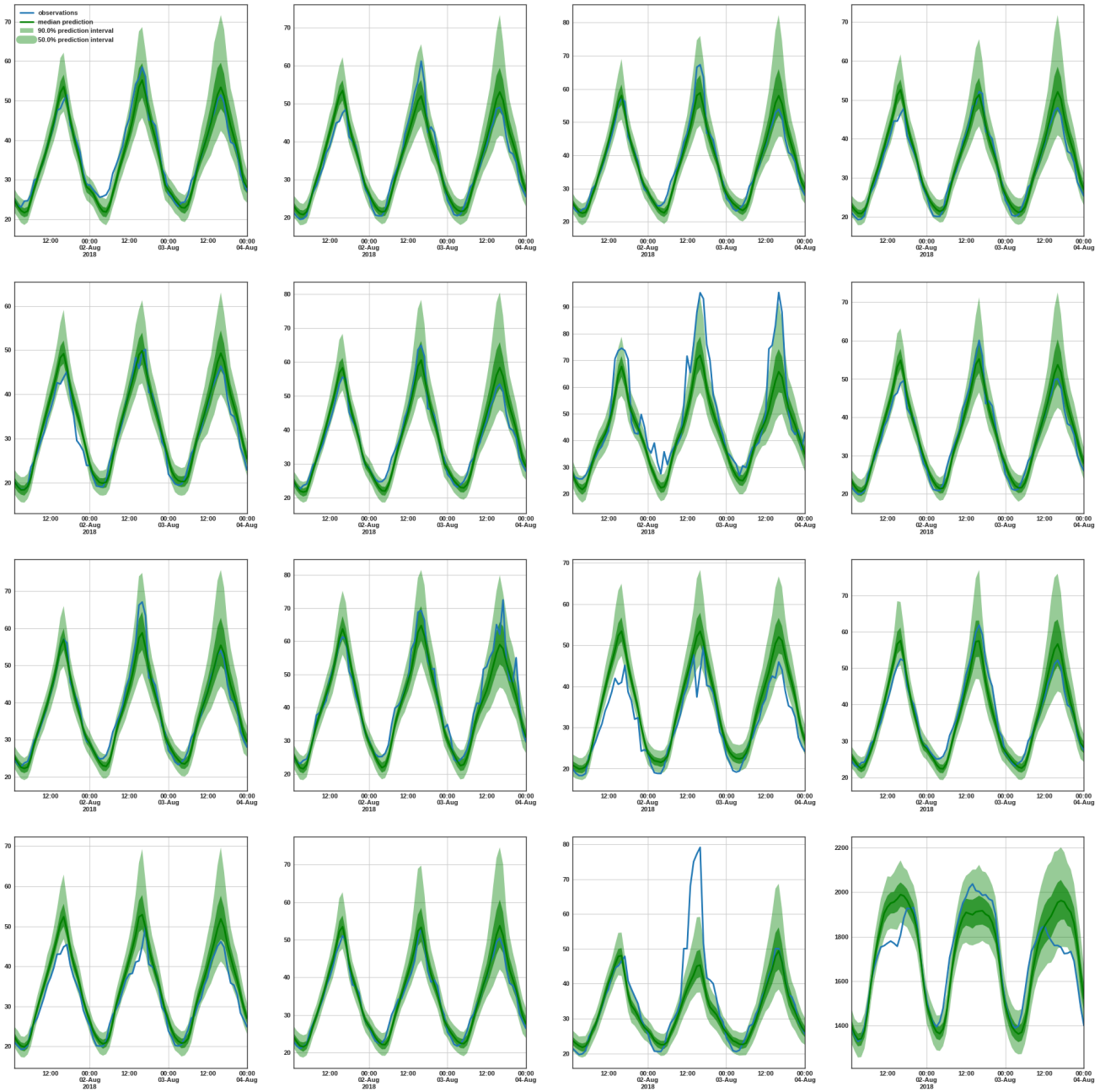


Figure 1.1: Ωριαίες πιθανοτικές προβλέψεις του TimeGrad για ορίζοντες 3 ημερών.

έναντι όλων των κλασικών προβλεπτών και μοντέλων βαθιάς μάθησης (AR, ARIMA, SARIMA, VAR, CNN, LSTM) σε όλες τις μετρικές αξιολόγησης (MSE, RMSE, MAE, MAPE, CRPS), όπως φαίνεται και στην Εικόνα 1.2. Σημειώνουμε πως το TimeGrad παρέχει πιθανοτικές προβλέψεις αντί για σημειακές, προσφέροντας πολύτιμη πληροφορία για την αβεβαιότητα των προβλέψεων.

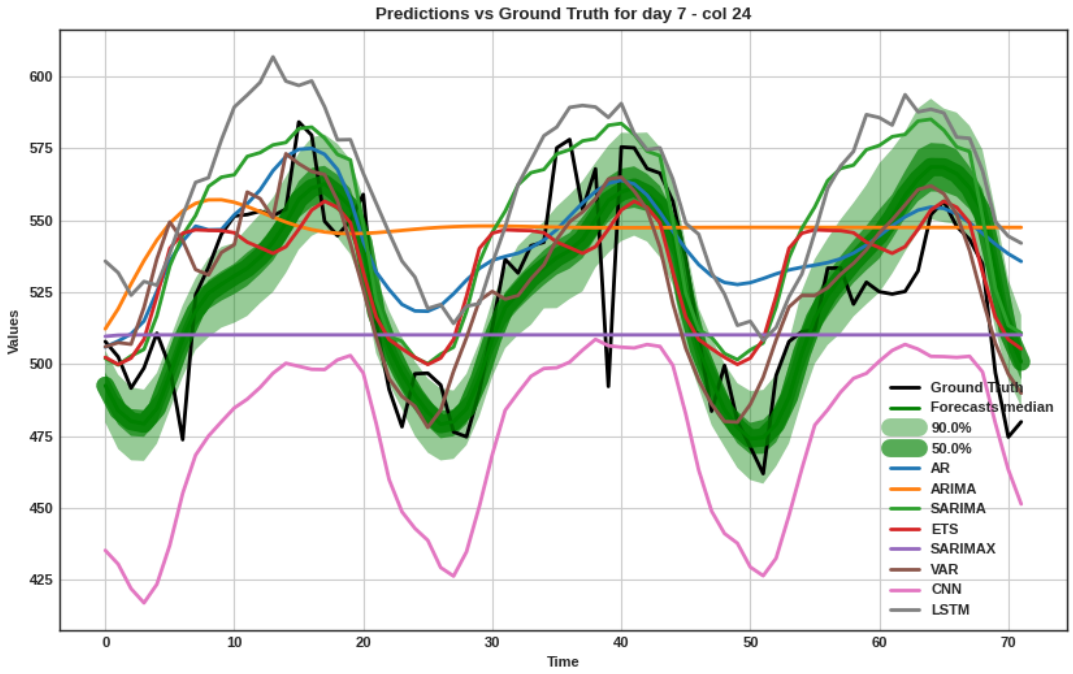


Figure 1.2: Σύγκριση των επιδόσεων του TimeGrad σε ορίζοντα 3 ημερών με άλλες κλασικές μεθόδους και τις πραγματικές τιμές ενέργειας σε ένα σχετικά θορυβώδες τμήμα των δεδομένων.

1.4.2 Περιβάλλον Προσομοίωσης

Το περιβάλλον προσομοίωσης του πράκτορα ορίστηκε ως μια Μερικώς Παρατηρήσιμη Διαδικασία Αποφάσεων Markov (POMDP), ώστε να μοντελοποιήσει τη λειτουργία ενός Συστήματος Αποθήκευσης Ενέργειας με Μπαταρίες (BESS) στο δίκτυο ηλεκτρικής ενέργειας. Το TimeGrad χρησιμοποιείται για την παροχή προβλεπόμενων τιμών σε ορίζοντα τριών ημερών.

Σε κάθε χρονικό βήμα k, το περιβάλλον βρίσκεται σε μια κρυφή κατάσταση s_k και δίνει μια παρατήρηση $o_k = \langle o_k^d, o_k^s \rangle$, όπου:

- o_k^d: Η ντετερμινιστική συνιστώσα, που είναι η κατάσταση φόρτισης της μπαταρίας (State of Charge - SoC), και εξελίσσεται σύμφωνα με γνωστή δυναμική.
- o_k^s: Η στοχαστική συνιστώσα, που αποτελείται από τις τιμές ηλεκτρικής ενέργειας, των οποίων η δυναμική επηρεάζεται από αβέβαιους παράγοντες της αγοράς.

Βάσει των παρατηρούμενων πληροφοριών, ο πράκτορας επιλέγει μια δράση $a_k \in \mathcal{A}$. Η παρατήρηση μεταβαίνει ως εξής:

$$\begin{aligned} \text{SoC}_{k+1} &= \text{SoC}_k + \eta\, a_k, \quad \text{με } \text{SoC}_{k+1} \in [0,1], \\ p_{k+1} &= \text{Prices}(k+1), \end{aligned}$$

όπου $\eta = 0.95$ είναι η απόδοση φόρτισης/εκφόρτισης της μπαταρίας. Ο χώρος δράσεων a_k ορίζεται από τα λειτουργικά όρια της μπαταρίας:

$$a_k \in \left[-\min\left\{ u_{max}, \frac{\text{SoC}_k}{\eta} \right\}, \min\left\{ u_{max}, \frac{1-\text{SoC}_k}{\eta} \right\} \right]$$

όπου $u_{max} = 0.25$ είναι η μέγιστη ροή ενέργειας. Η μπαταρία έχει συνολική χωρητικότητα 192.000 Wh.

Η ανταμοιβή σε κάθε χρονικό βήμα υπολογίζεται ως:

$$R(o_k, a_k) = \text{Revenue}(o_k, a_k) - \text{Degradation}(o_k, a_k)$$

όπου το Revenue προκύπτει από την αγορά ή πώληση ενέργειας στην παρατηρούμενη τιμή, και το κόστος Degradation υπολογίζεται με βάση το Βάθος Εκφόρτισης (Depth-of-Discharge - DoD) και τη διάρκεια ζωής της μπαταρίας. Το μοντέλο της μπαταρίας είναι εμπνευσμένο από την εργασία [41].

Πειραματικά Αποτελέσματα του Προτεινόμενου Αλγορίθμου D-I MPC

Ο αλγόριθμος Diffusion-Informed MPC (D-I MPC), ο οποίος ενσωματώνει το TimeGrad σε αλγορίθμους MPC, αξιολογήθηκε εκτενώς στο περιβάλλον για τη διαχείριση ενός συστήματος αποθήκευσης ενέργειας μπαταρίας (BESS) στην αγορά ηλεκτρικής ενέργειας.

Προβλεπτικός Έλεγχος (MPC) Ο ντετερμινιστικός MPC με TimeGrad χρησιμοποιεί μια προβλεπόμενη τροχιά τιμών για να βελτιστοποιήσει τις ενέργειες του BESS (φόρτιση/εκφόρτιση) σε έναν πεπερασμένο ορίζοντα. Ο αλγόριθμος επανασχεδιάζει τις ενέργειες καθημερινά, για να προσαρμόζεται στις νέες τιμές και προβλέψεις. Ένα στιγμιότυπο βελτιστοποίησης του αλγορίθμου φαίνεται στην Εικόνα 1.3. Τα βασικά συμπεράσματα από τα πειράματα που εκτελέσαμε είναι:

- Επίτευξη Στόχου: Το BESS αγοράζει ενέργεια όταν οι τιμές είναι χαμηλές και πουλάει όταν είναι υψηλές, οδηγώντας σε κέρδος.
- Περιορισμοί Φόρτισης/Εκφόρτισης: Το σύστημα κατανέμει τη φόρτιση και την εκφόρτιση σε πολλαπλά βήματα λόγω των φυσικών περιορισμών της μπαταρίας.
- Βραχυπρόθεσμη Βελτιστοποίηση: Ένας βασικός περιορισμός, όπως αναφέρθηκε και στις προηγούμενες ενότητες, είναι ότι το MPC τείνει να αποφορτίζει πλήρως την μπαταρία στο τέλος του ορίζοντα για να μεγιστοποιήσει το άμεσο κέρδος, αγνοώντας το μακροπρόθεσμο κέρδος (μετά το πέρας του ορίζοντα).

Στοχαστικός Προβλεπτικός Έλεγχος (Stochastic MPC) Ο Στοχαστικός MPC (SMPC) ενσωματώνει την αβεβαιότητα απευθείας στη βελτιστοποίηση, χρησιμοποιώντας την κατανομή πιθανότητας των μελλοντικών τιμών ενέργειας. Για μια αρχική υλοποίηση, χρησιμοποιήθηκε μία μόνο τροχιά από την κατανομή προβλέψεων του TimeGrad. Ένα παράδειγμα βελτιστοποίησης του αλγορίθμου φαίνεται στην Εικόνα 1.4. Παραθέτουμε τα κύρια αποτελέσματα από τα πειράματα:

- Αντιμετώπιση Αβεβαιότητας: Ο SMPC λαμβάνει υπόψη τη στοχαστική φύση των μελλοντικών τιμών.
- Υποβέλτιστες Ενέργειες: Η χρήση μόνο μιας τροχιάς οδηγεί σε υποβέλτιστες ενέργειες, καθώς το μοντέλο ερμηνεύει το θόρυβο ως ευκαιρία κέρδους. Αυτό εκδηλώνεται με συχνές αποφάσεις φόρτισης/εκφόρτισης που δεν είναι λογικές.
- Συνέπεια TimeGrad: Παρά τον θόρυβο, η ακρίβεια του TimeGrad επιτρέπει στο SMPC να παράγει μια λογική ακολουθία ενεργειών, καθιστώντας τον αλγόριθμο χρήσιμο όταν η παραγωγή πολλαπλών τροχιών δεν είναι εφικτή.

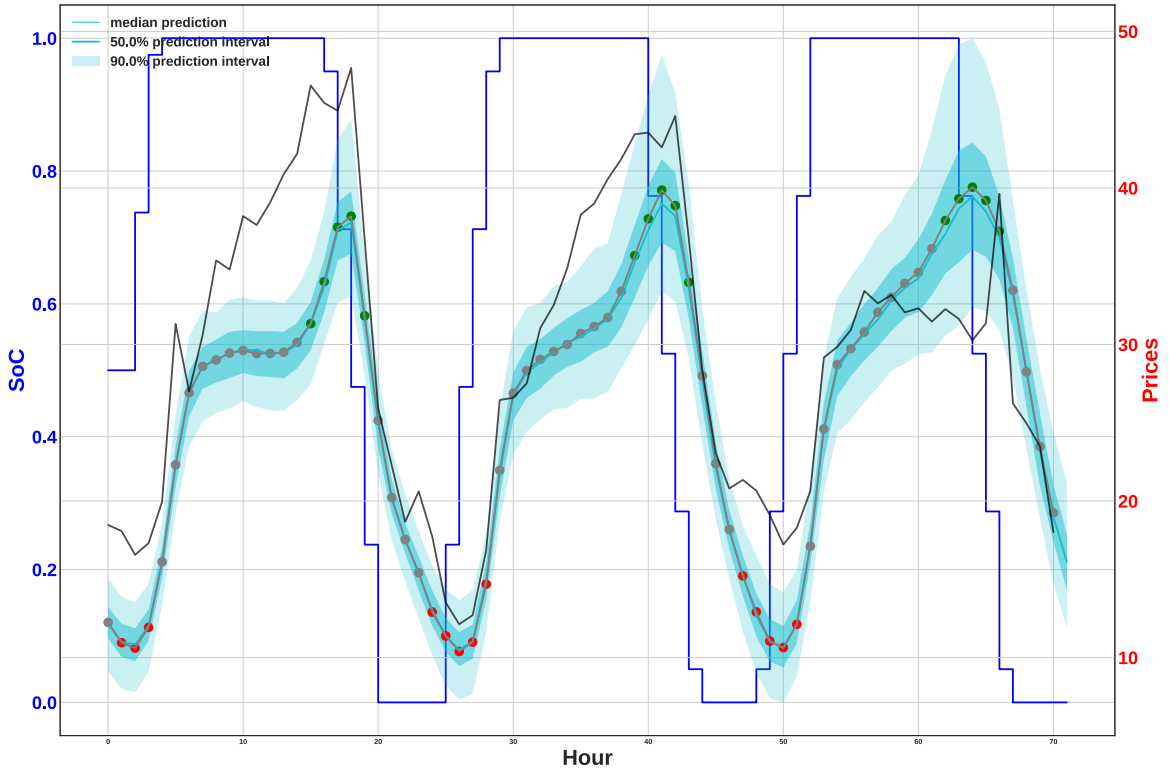


Figure 1.3: Στρατηγική δράσεων που σχεδιάζει ο αλγόριθμος MPC για ορίζοντα 3 ημερών.

- Πεπερασμένος Ορίζοντας: Όπως και το ντετερμινιστικό MPC, το SMPC οδηγεί σε πλήρη εκφόρτιση της μπαταρίας στο τέλος του ορίζοντα, αγνοώντας το κέρδος μετά το πέρας του ορίζοντα.

Προβλεπτικός Έλεγχος με Προσομοιώσεις Motne-Carlo (Monte-Carlo SMPC) Το MC-SMPC παρέχει εύρωστο έλεγχο, λαμβάνοντας υπόψη την αβεβαιότητα της πρόβλεψης στη διαδικασία βελτιστοποίησης, χρησιμοποιώντας πολλαπλές τροχιές τιμών από την κατανομή του TimeGrad. Ένα βήμα βελτιστοποίησης του αλγορίθμου δίνεται στην Εικόνα 1.5. Τα βασικά συμπεράσματα των πειραμάτων έδειξαν τα εξής:

- Ενισχυμένη Ευρωστία: Βελτιστοποιώντας σε πολλές τροχιές του μέλλοντος (π.χ., 100), το MC-SMPC σχεδιάζει δράσεις που είναι πιο ανθεκτικές.
- Μείωση Θορύβου: Μετριάζει το πρόβλημα των θορυβωδών προβλέψεων, καθώς βελτιστοποιώντας την αναμενόμενη τιμή (εκτιμώμενη από τον μέσο όρο) ο θόρυβος αυτοαναιρείται, οδηγώντας σε πιο σταθερές στρατηγικές.
- Υπολογιστικό Κόστος: Το τίμημα της χρήστης αυτής της εύρωστης μεθόδου είναι η αυξημένη υπολογιστική πολυπλοκότητα.
- Περιορισμός Πεπερασμένου Ορίζοντα: Παρόμοια με τις προηγούμενες μεθόδους, ο περιορισμός του ορίζοντα βελτιστοποίησης παραμένει άλυτος.

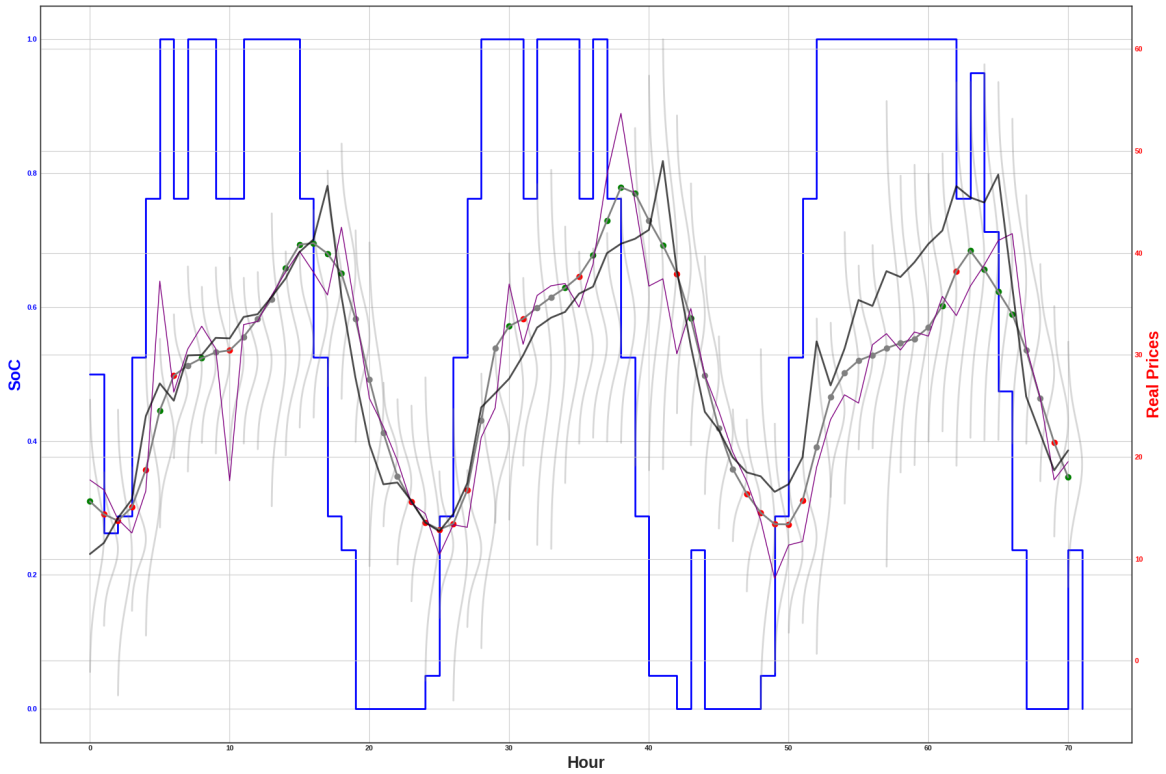


Figure 1.4: Στρατηγική δράσεων που σχεδιάζει ο αλγόριθμος SMPC για ορίζοντα 3 ημερών.

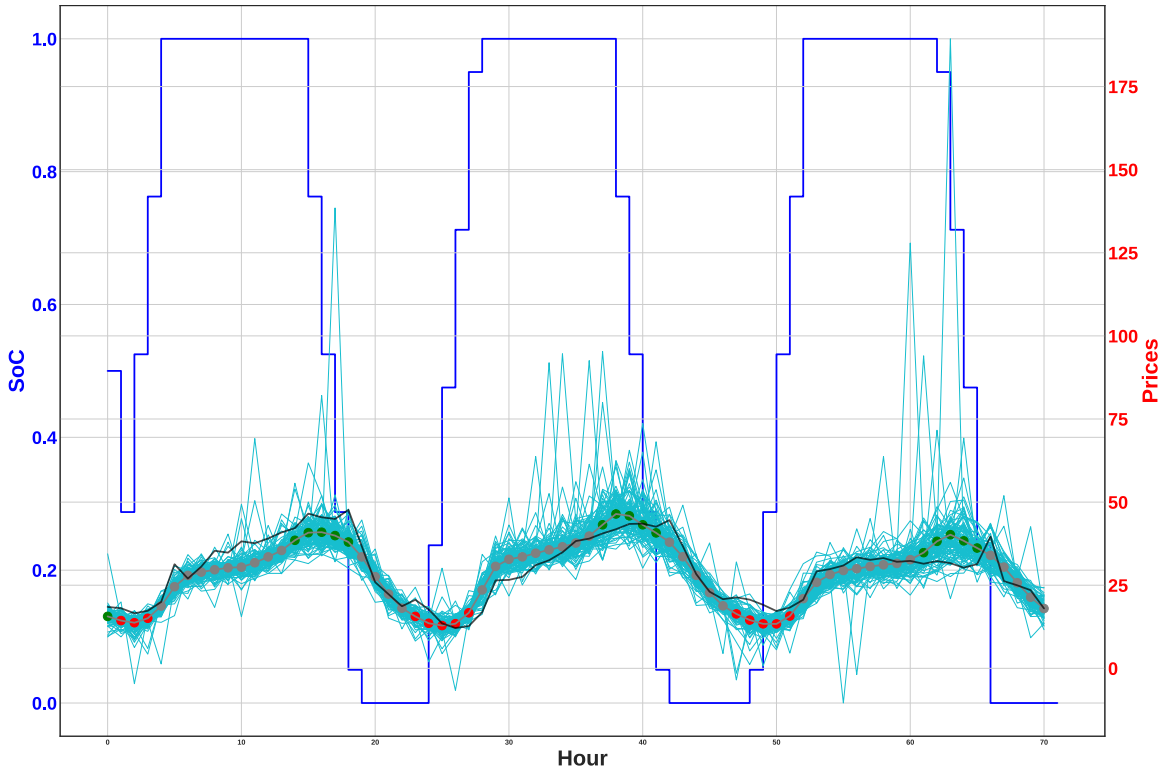


Figure 1.5: Στρατηγική δράσεων που σχεδιάζει ο αλγόριθμος Monte Carlo SMPC για ορίζοντα 3 ημερών.

Προβλεπτικός Έλεγχος Βασισμένος σε Δέντρα Σεναρίων (Scenario Tree-Based MPC) Ο MPC που βασίζεται σε Δέντρο Σεναρίων προσφέρει μια δομημένη προσέγγιση για τη διαχείριση της αβεβαιότητας, διακριτοποιώντας την κατανομή των μελλοντικών τιμών σε ένα δέντρο. Ένα στιγμιότυπο της βελτιστοποίησης φαίνεται στην Εικόνα 1.6. Τα αποτελέσματα των πειραμάτων έδειξαν τα εξής:

- Non-anticipativity: Οι αποφάσεις ελέγχου σε κάθε κόμβο του δέντρου λαμβάνονται μόνο με βάση τους προγόνους του, δηλαδή με την πληροφορία που έχει πραγματοποιηθεί ως την τρέχουσα στιγμή.
- Βελτιστοποίηση Αναμενόμενου Κέρδους: Ο αλγόριθμος βελτιστοποιεί το σταθμισμένο ως προς την πιθανότητα άθροισμα των άμεσων ανταμοιβών σε όλους τους κόμβους του δέντρου.
- Εύρωστη Στρατηγική: Και οι δύο παραλλαγές του αλγορίθμου παράγουν αποδοτική στρατηγική για τον πράκτορα.
- Περιορισμός Πεπερασμένου Ορίζοντα: Τα μοντέλα δεν δύνανται να βελτιστοποιήσουν για βήματα μετά το πέρας του ορίζοντα.

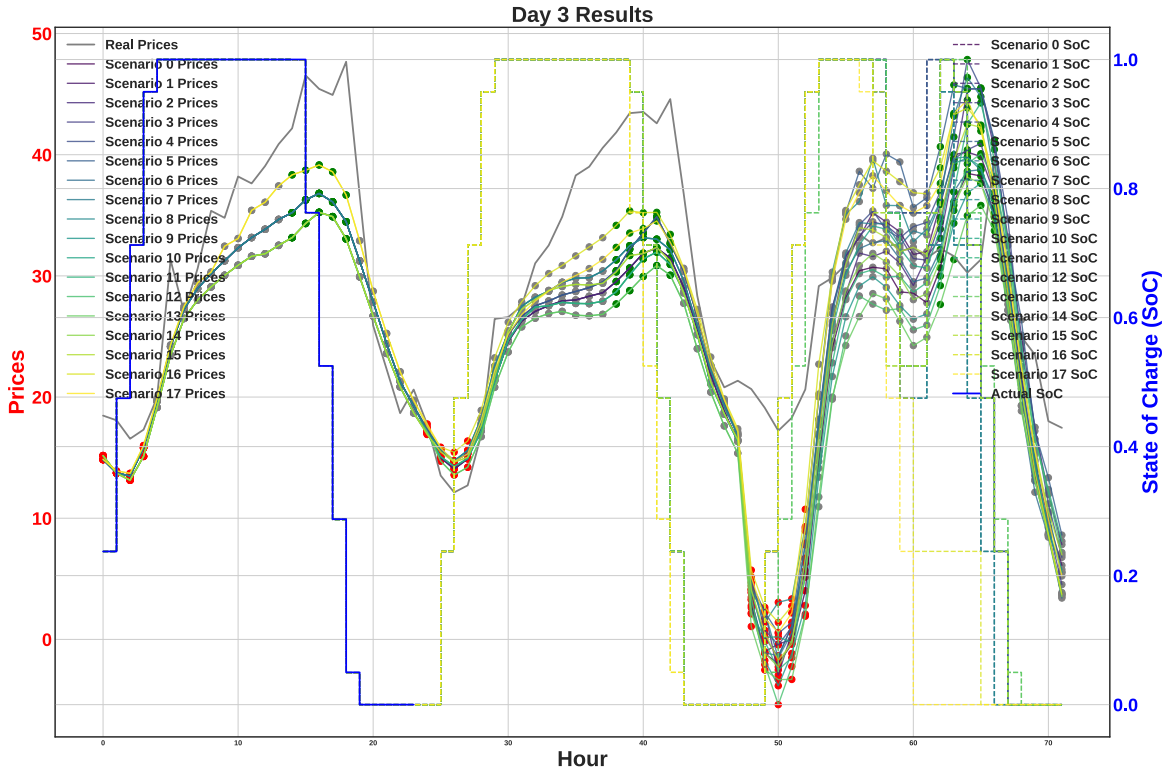


Figure 1.6: Στρατηγική δράσεων που σχεδιάζει ο αλγόριθμος Forward-Clustering Scenario Tree-Based MPC για ορίζοντα 3 ημερών.

Προβλεπτικός Έλεγχος Ενισχυμένος με Ευρετική (Heuristic-Augmented MPC) Ο H-A MPC χρησιμοποιεί μια ευρετική συνάρτηση στο τέλος του ορίζοντα πρόβλεψης, η οποία εκτιμά τη βέλτιστη τερματική κατάσταση. Όπως φαίνεται και στην Εικόνα 1.7 η βέλτιστη ακολουθία εισόδων δεν καταλήγει στην πώληση όλης της ενέργειας στο πέρας του ορίζοντα, αλλά διατηρείται κάποιο φορτίο καθώς αναμένεται ότι θα είναι χρήσιμο μελλοντικά.

- Απλή Ευρετική: Αντί να χρησιμοποιήσουμε μια κοινή ευρετική η οποία θα χρειαζόταν πολλά δεδομένα για να εκτιμήσει το αναμενόμενο κέρδος κάθε δυνατής τερματικής κατάστασης, εκτιμάμε μόνο τη βέλτιστη τελική κατάσταση.
- Το LSTM που προβλέπει την βέλτιστης τελική κατάσταση έχει μεγάλη ακρίβεια.
- Συνεπής Βελτίωση: Το H-A MPC βελτιώνει σταθερά, αν και ελαφρώς, τη διαδικασία βελτιστοποίησης του τυπικού MPC, αποδεικνύοντας ότι έχει τη δυνατότητα να βελτιώσει τον MPC πιθανώς περισσότερο σε άλλες εφαρμογές (με μικρότερο ορίζοντα ή περισσότερο θόρυβο).

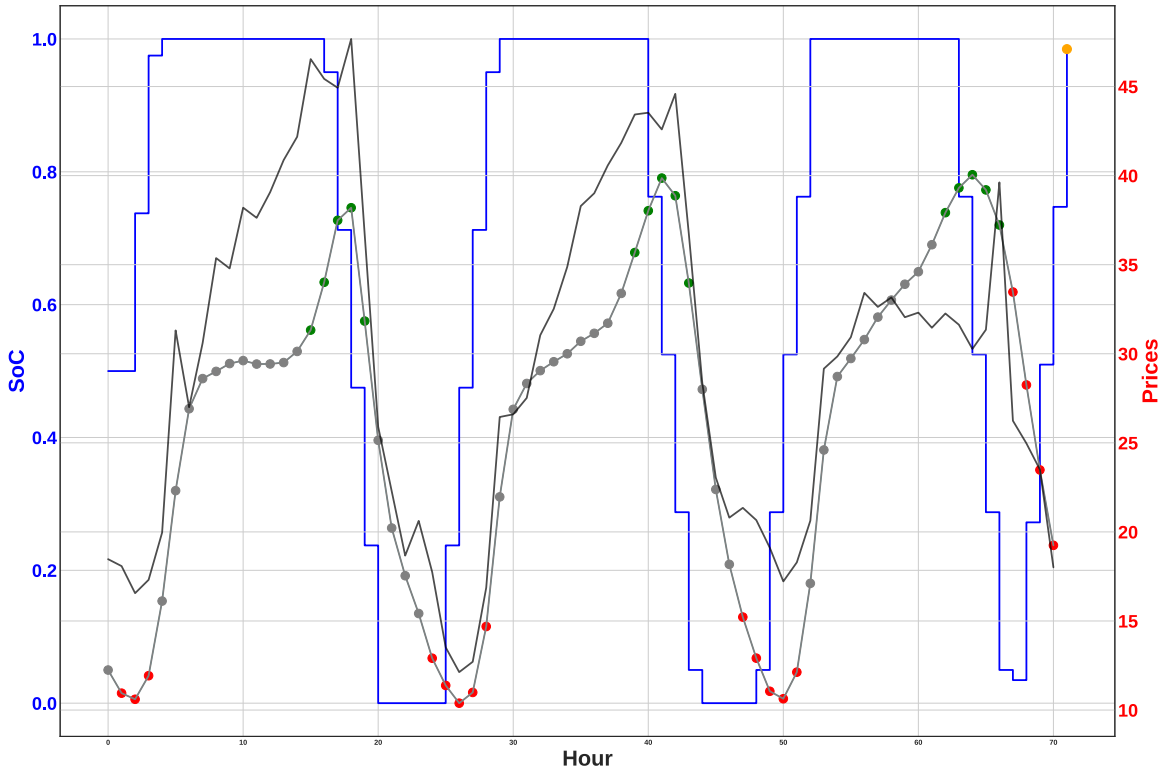


Figure 1.7: Στρατηγική δράσεων που σχεδιάζει ο αλγόριθμος Heuristic-Augmented MPC Optimizer για ορίζοντα 3 ημερών.

1.4.3 Σύγκριση με Κλασσικές Μεθόδους Πρόβλεψης

Για την περαιτέρω αξιολόγηση της προσέγγισής μας (Diffusion-Informed MPC) τη συγκρίναμε έναντι αντίστοιχων υλοποιήσεων MPC καθοδηγούμενες όμως από κλασικά μοντέλα πρόβλεψης (AR, ARIMA, SARIMA, VAR, CNN, LSTM) και παρατηρήσαμε πως η μέθοδός μας δίνει σημαντικά καλύτερα αποτελέσματα.

- Υπεροχή *D-I MPC*: ο αλγόριθμος *D-I MPC* προσφέρει πολύ πιο σθεναρό έλεγχο από όλες τις κλασσικές μεθόδους πρόβλεψης που δοκιμάσαμε, με πλεονέκτημα 38.8% έναντι της καλύτερης κλασσικής μεθόδου, ως προς τη μέση ανταμοιβή στο περιβάλλον που δοκιμάστηκαν.

- Αναποτελεσματικότητα Μοντέλων Βαθιάς Μάθησης: Παρόλο που τα μοντέλα βαθιάς μάθησης (CNN, LSTM) είχαν καλύτερες μετρικές σφάλματος πρόβλεψης (prediction accuracy), η απόδοσή τους στην καθοδήγηση του MPC ήταν κατώτερη από τα απλά αυτοπαλινδρομικά μοντέλα. Αυτό οφείλεται στο γεγονός ότι στην προσπάθεια μείωσης του σφάλματος πρόβλεψης, τα μοντέλα βαθιάς μάθησης έμαθαν να προβλέπουν τα ακρότατα των χρονοσειρών τιμών ενέργειας με κάποιες χρονικές αποκλίσεις, οι οποίες είναι κρίσιμες για τη λήψη αποφάσεων στην ενεργειακή εξισορροπητική κερδοσκοπία. Αντιθέτως, λόγω του είδους της εφαρμογής, επειδή τα ακρότατα αυτά έχουν ισχυρή χρονική εξάρτηση, είναι πιο εύκολο να προβλεφθούν από αυτοπαλινδρομικά μοντέλα.

- Σημασία Λεπτομερούς Πρόβλεψης: Η ακριβής πρόβλεψη των λεπτομερειών στην τροχιά της κατάστασης, ιδίως των σημείων καμπής, έχει τον πιο σημαντικό ρόλο. Τα μοντέλα πρόβλεψης χρονοσειρών βασισμένα σε διαδικασίες διάχυσης, όπως το TimeGrad, έχουν την ικανότητα να παράγουν πολύ λεπτομερείς τροχιές, λόγω της εγγενούς πολυπλοκότητάς τους, και ως αποτέλεσμα είναι ιδανικά για την καθοδήγηση του MPC, οδηγώντας σε καλύτερο έλεγχο.

1.4.4 Σύγκριση με Προσεγγίσεις Ενισχυτικής Μάθησης

Η σύγκριση της προσέγγισής μας και με μεθόδους Ενισχυτικής Μάθησης Χωρίς Μοντέλο (model-free RL), αποδεικνύει ότι οι μέθοδοι μάθησης βασισμένες σε μοντέλο υπερέχουν σημαντικά όταν είναι διαθέσιμα ακριβή μοντέλα για το σύστημα.

- Σημαντική Υπεροχή *D-I MPC*: Η καλύτερη υλοποίηση model-free RL, με πλήρη γνώση των επόμενων 12 πραγματικών τιμών (μη-ρεαλιστικό σενάριο), απέδωσε 69.5% χειρότερα από τη δική μας μέθοδο. Αυτό υπογραμμίζει το πλεονέκτημα της χρήσης ενός καλού μοντέλου (όπως το TimeGrad) για τη βελτιστοποίηση των αποφάσεων.

- Ευρωστία και Ευκολία ρύθμισης: Η μέθοδός μας είναι πιο εύρωστη και πολύ ευκολότερη στη ρύθμιση (tuning). Μόλις βρεθούν καλές παράμετροι για το μοντέλο διάχυσης, οι προβλέψεις του παραμένουν συνεπείς, άρα η διαδικασία εκπαίδευσης είναι πολύ πιο εύκολη και σταθερή.

- Προκλήσεις Model-Free RL: Η εκπαίδευση είναι εξαιρετικά δύσκολη, και απαιτεί τεράστιο αριθμό επαναλήψεων σε σύνθετες εφαρμογές (στην περίπτωσή μας χρειαστήκαμε 15.000.000 επαναλήψεις) λόγω του μεγάλου χώρου παραμέτρων.

- Πρακτικά Ζητήματα: Ενώ στη θεωρία οι μέθοδοι RL Χωρίς Μοντέλο μπορούν να προσεγγίσουν τη βέλτιστη συμπεριφορά μετά από αρκετές επαναλήψεις, το πρακτικό κόστος σε χρόνο και υπολογιστικούς πόρους είναι μεγάλο. Σε σενάρια όπου ένα ακριβές μοντέλο συστήματος είναι διαθέσιμο, οι προσεγγίσεις που βασίζονται σε μοντέλα προσφέρουν πολύ πιο αποδοτική συμπεριφοράς για τον πράκτορα.

1.5 Συμπεράσματα και Μελλοντική Εργασία

1.5.1 Συμπεράσματα

Στην παρούσα Διπλωματική Εργασία αναπτύξαμε καινοτόμες προσεγγίσης για τη λήψη αποφάσεων υπό αβεβαιότητα σε στοχαστικά συστήματα. Οι προσεγγίσεις αυτές ενσωματώνουν πιθανοτικά

μοντέλα πρόβλεψης βασισμένα σε διαδικασίες διάχυσης σε αλγορίθμους Προβλεπτικού Ελέγχου (MPC), με σκοπό τον έλεγχο σύνθετων πολυδιάστατων στοχαστικών συστημάτων.

Μέσα από εκτεταμένα πειράματα, δείξαμε ότι η μέθοδός μας, *Diffusion-Informed MPC*, επιτυγχάνει σημαντικά καλύτερες επιδόσεις σε σύγκριση με παραδοσιακές προσεγγίσεις. Αυτό οφείλεται στην εγγενή ικανότητα των μοντέλων διάχυσης να μαθαίνουν με ακρίβεια περίπλοκες κατανομές πιθανότητας. Επιπλέον, οι παραλλαγές του αλγορίθμου μας προσέγγισαν πολύ τις ιδανικές υλοποιήσεις (χρησιμοποιώντας τέλεια μοντέλα πρόβλεψης), υποδεικνύοντας ότι η μέθοδός μας βελτιστοποίησε τον έλεγχο του πράκτορα σχεδόν τόσο αποτελεσματικά όσο οι ιδανικοί αλγόριθμοι.

Επιπλέον, η μέθοδός μας, επέδειξε πλεονέκτημα έναντι διαφόρων υλοποιήσεων RL Χωρίς Μοντέλο. Η διαφορά αυτή υπογραμμίζει τα οφέλη της χρήσης ενός καλού μοντέλου πρόβλεψης για βελτιστοποίηση όταν αυτό είναι διαθέσιμο. Ένα ακόμα πλεονέκτημα της προσέγγισής μας είναι η ευρωστία και η ευκολότερη ρύθμιση, σε αντίθεση με τη χρονοβόρα εκπαίδευση και εύρεση παραμέτρων των μεθόδων RL Χωρίς Μοντέλο.

1.5.2 Μελλοντική Εργασία

Μερικές μελλοντικές κατευθύνσεις που πηγάζουν από την έρευνά μας είναι:

1. Εξυπνότερο Κλάδεμα Δέντρου Σεναρίων: Παρόλο που τα Δέντρα Σεναρίων Διάχυσης οδηγούν σε μια σθεναρή διαδικασία βελτιστοποίησης, μπορεί να γίνει περαιτέρω έρευνα για τον μετριασμό των επιπτώσεων του κλαδέματος του δέντρου. Αυτό σημαίνει την εξερεύνηση μεθόδων κλαδέματος (πιθανώς με χρήση ευρετικών) που προσαρμόζονται δυναμικά στα παραγόμενα σενάρια.

2. Υβριδικές Αρχιτεκτονικές Ενισχυτικής Μάθησης: Μια άλλη μελλοντική κατεύθυνση περιλαμβάνει τη δημιουργία υβριδικών προσεγγίσεων RL που συνδυάζουν τα πλεονεκτήματα τόσο των τεχνικών χωρίς μοντέλο όσο και των τεχνικών που βασίζονται σε μοντέλο, όπως Dyna-Style μεθόδοι, όπου ένα μοντέλο χρησιμοποιείται για τον εμπλουτισμό των εμπειρικών για εκπαίδευση, ή των Q-learning τεχνικών που χρησιμοποιούν ένα μοντέλο για την εκτίμηση του κόστους μετά τον ορίζοντα (cost-to-go).

3. Εφαρμογή και σε άλλα Περιβάλλοντα: Για περισσότερες ενδείξεις της αποτελεσματικότητας και της μεθόδου μας, μπορούμε να εξερευνήσουμε επιπλέον περιπτώσεις εφαρμογής, όπως ο έλεγχος έξυπνων δικτύων (smart grid), η αυτόνομη οδήγηση ή η βελτιστοποίηση χρηματοοικονομικών χαρτοφυλακίων (financial portfolio optimization).

4. Σύγκριση Διαφόρων Μοντέλων Διάχυσης σε MPC: Ενώ το TimeGrad παρείχε ένα ισχυρό μοντέλο πρόβλεψης, θα ήταν ενδιαφέρον να εξερευνηθεί η απόδοση και άλλων μοντέλων διάχυσης για χρονοσειρές.

5. Ενσωμάτωση Ανίχνευσης Ανωμαλιών (Anomaly Detection): Σε μελλοντική έρευνα θα μπορούσαμε επίσης να διερευνήσουμε την ενσωμάτωση μοντέλων διάχυσης για την ανίχνευση ανωμαλιών. Αυτό πιθανώς θα οδηγήσει σε πιο ακριβή σχεδιασμό.

6. Ενσωμάτωση Δεδομένων Γεγονότων (Event Data): Ένα ακόμα μελλοντικό βήμα είναι η συμπερίληψη και άλλων πηγών δεδομένων, όπως πληροφορίες καιρού και ειδήσεις. Αυτοί οι εξωτερικοί παράγοντες παρέχουν πολύτιμες πληροφορίες σχετικά με τις απρόβλεπτες συμπεριφορές του συστήματος και μπορούν να προστεθούν στο μοντέλο διάχυσης με τη μορφή διανυσμάτων-συνθηκών (condition-vectors or covariates).

Chapter 2

Introduction

Contents

2.1 The Evolution of Artificial Intelligence and Machine Learning

The concept of creating intelligent machines has ancient roots, dating back thousands of years to philosophers who explored the nature of intelligence and consciousness. This human interest materialized in early autonomous machines known as automatons, a term derived from ancient Greek, meaning "acting of one's own will". Greek mythology has a plethora of tales about mechanical beings, such as the bronze giant Talos, who patrolled the island of Crete, and the self-moving golden tripods crafted by Hephaestus to serve the gods. Moving from the tales to engineering, the philosopher Archytas of Tarentum is known to have designed a wooden, steam-powered pigeon capable of flight, in the 4th century BCE. Philo of Byzantium created the "Automate Therapaenis" (Automatic Maid), in 3rd century BCE, a human-sized automaton that could automatically pour wine and water from two containers. More popular inventions are credited to Heron of Alexandria in the 1st century CE, with various works, such as Aeolipile (a steam-powered rotating spheres), automatic doors, and even robotic theatrical plays that were mimicking humans.

The formal establishment of Artificial Intelligence (AI) as a field of study occurred two millennia later, in 1956, at the Dartmouth Summer Research Project on Artificial Intelligence. This conference is widely regarded as the birth of AI. It was during this event that McCarthy invented the term "Artificial Intelligence". Early AI research was mainly centered on symbolic methods and problem-solving algorithms, aiming to mimic human reasoning through *explicit rules*. Figure 2.1 depicts the pioneers gathered at this event.

Figure 2.1: In the back row from left to right are Oliver Selfridge, Nathaniel Rochester, Marvin Minsky, and John McCarthy. In front on the left is Ray Solomonoff; on the right, Claude Shannon. The identity of the person between Solomonoff and Shannon remained a mystery for some time. Source: [1]

Since its beginning, AI has evolved rapidly. Researchers, recognizing the limitations of such rule-based systems in handling the *variability and uncertainty* of real-world scenarios, developed Machine Learning (ML) and Deep Learning (DL), which enables systems to learn from data rather than being explicitly programmed. Examples of early ML applications are Arthur Samuel's checkers-playing program [42] in 1959 and Frank Rosenblatt's Perceptron [10] in 1958. However, these early models had limitations in handling complex, non-linear problems, leading to a period of reduced interest.

Neural networks gained interest again in the 1980s and 1990s, thanks to theoretical advancements and increase in computational power. A pivotal moment happened in 2012, when the deep convolutional neural network AlexNet showcased state-of-the-art performance in image classification [43]. Even more impressive breakthroughs took place in the subsequent decade, with the rise of Large Language Models (LLMs) that revolutionized natural language generation and understanding, and the rise of Diffusion Models which show remarkable capabilities in generating high-quality data.

Today, AI, ML, and DL are integral parts of various everyday applications, from well-known examples like facial recognition, natural language processing, and recommendation systems, to *intelligent decision-making* in many problems. In real-world scenarios, which usually involve complex stochastic systems, like energy grids or financial markets, AI has the ability to process large datasets and identify multi-factor hidden patterns which are impossible to be captured by human intuition or traditional modeling. This has led to great applications in time series forecasting, where models are trained to predict future values based on historical data, and in system optimization, with Reinforcement Learning (RL), where intelligent agents learn to make efficient sequential decisions to achieve a goal.

2.2 Forecasting and Decision-Making Under Uncertainty

In real-world systems, the underlying dynamics are complex and uncertain, therefore the future evolution is influenced mostly by stochastic factors. Applications such as energy management, financial trading, and autonomous systems require agents to make sequential decisions based on partial and noisy observations, without full knowledge of the system dynamics.

Traditional Time Series Forecasting Traditional time series forecasting models, such as autoregressive models, have been widely used, with some success. However, their inherent simplicity (for example, they often assume linearity and stationarity), limits their effectiveness in capturing the complex patterns of real-world data. Machine Learning models like Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, utilize the complexity of neural networks to improve the ability of modeling the intricate patterns. However, they typically yield point forecasts, since they are trained using the MSE loss, and do not account for the uncertainty. To address this, one can output quantiles or use Bayesian LSTMs (for example [44]), but they still struggle with multi-modal futures and complex dependencies.

More recently, diffusion models have emerged and showcase incredible potential in modeling complex data distributions. This makes them highly suitable to probabilistically forecast time series, leading to even further improvements in accuracy.

Reinforcement Learning in Partially Observable Environments Reinforcement Learning (RL) provides a framework for agents to learn optimal policies through interactions with

an environment, in order to achieve a goal. In the real-world, the environment is partially observable, meaning that the agent must make decisions based on incomplete information.

Model-Free Reinforcement Learning (MFRL), directly learns from experience without creating an explicit model the environment's dynamics. The primary advantage of this learning method lies in simplicity and applicability, as MFRL algorithms do not require prior knowledge, or the computational burden of building an accurate model of the environment dynamics. However, a big drawback is that they usually require huge numbers of training interactions to converge to effective behaviors.

Model-Based Reinforcement Learning (MBRL) enhances the efficiency of RL by incorporating a step of learning a model of the environment's dynamics. These models can then be used for planning, reducing the complexity. However, accurately modeling the dynamics is a significant challenge, especially in high-dimensional and stochastic settings.

Model Predictive Control is a receding-horizon control algorithm that uses a predictive model of the system to optimize a sequence of future control inputs at each time step. Only the first input in the optimized sequence is applied before the process repeats using new observations, enabling adaptive, feedback-based planning in partially observable and uncertain environments.

Probabilistic Forecasting in Decision-Making The integration of advanced probabilistic forecasting models with decision-making frameworks can significantly improve performance in uncertain environments. Using state-of-the-art diffusion models for time series forecasting, agents can generate realistic scenarios in order to plan their actions in the presence of uncertainty. Incorporating these probabilistic forecasts into control and planning algorithms enhances *robustness*, provides *risk-aware decisions*, and increases training efficiency by utilizing the generated trajectories.

2.3 Contributions of This Thesis

This thesis explores the integration of diffusion-based probabilistic forecasting models with sequential decision-making frameworks to provide a robust and efficient model for an agent in partially observable stochastic environments. The main contributions are:

1. Adapting diffusion models for time series forecasting in partially observable stochastic dynamical systems to provide probabilistic predictions.

2. Developing deterministic and stochastic optimization algorithms that utilize sampled trajectories from the diffusion-based forecaster for robust control.

3. Proposing a scenario tree-based optimization algorithm to optimize actions over a structure that hierarchically organizes the uncertainty, having potential for risk-aware decision-making applications.

4. Formulating an algorithm that introduces a heuristic to implicitly extend the end of the optimization horizon, for more long-term planning.

5. Demonstrating the effectiveness of the proposed frameworks through a case study in energy arbitrage, showcasing improvement over using classical forecasting, and model-free RL algorithms.

6. Part of our research, has been accepted for publication at the EUSIPCO 2025 [9].

2.3.1 Organization of the Thesis

Chapter 3 provides a survey of fundamental works and models in deep learning, time series forecasting, generative models, diffusion models, and reinforcement learning.
Chapter 4 formalizes the class of problems that this thesis targets and defines the proposed theoretical frameworks for integrating diffusion forecasting with model-based RL.
Chapter 5 presents experimental results and evaluations of our methods against alternative approaches in the case study of energy arbitrage.
Chapter 6 summarizes the key findings the thesis, and discusses directions for future research.

Figure 2.2 presents an overview of the major ML areas covered in this thesis: from deep learning, through generative modeling, to reinforcement learning and its use in sequential decision-making. Finally, our approach combines diffusion-based probabilistic forecasting with sequential decision-making algorithms.

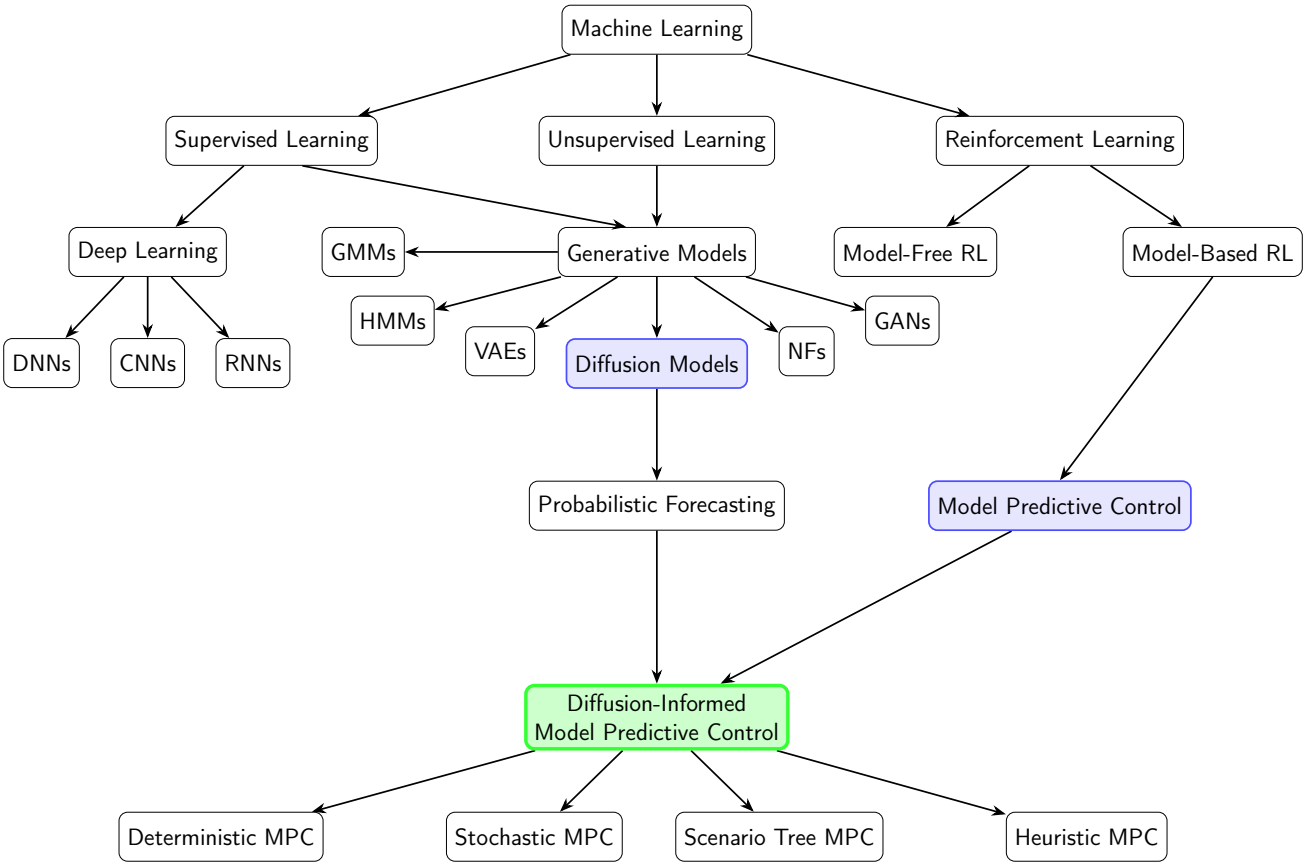


Figure 2.2: Overview of the Machine-Learning areas discussed in this thesis ending at the method we propose.

Chapter 3

Background and Related Work

Contents

3.1 Deep Learning

Deep learning is a subset of machine learning that uses deep neural networks (neural networks with many layers) to model complex patterns in data. These neural networks, often referred to as deep neural networks (DNNs), are composed of multiple layers of interconnected nodes, called neurons, that process information for various tasks by learning from large amounts of data. Deep learning aims to mimic how the human brain processes information. Deep neural networks are composed of sequences of linear mappings and nonlinear activation functions. During forward propagation, each layer transforms its input into a new representation, while during backpropagation, the network computes gradients of a loss function with respect to every weight, and updates them, in order to perform better in the given task.

3.1.1 Perceptron and Deep Neural Networks

The perceptron, introduced by Frank Rosenblatt in 1958 [10], is one of the earliest models of a neural network. It serves as the foundation for more complex architectures, including modern deep learning models. A perceptron consists of a single layer of neurons, each of which computes a weighted sum of inputs and applies an activation function to determine the output.

Inspired by the biological structure of the human brain, neural networks consist of interconnected layers of neurons (nodes). Each neuron applies an activation function to its input to produce an output for the successive neurons. The most common type of neural networks is feedforward networks, where data flows in one direction, from the input layer to the output layer, meaning that neurons in one layer do not connect to neurons in the same or previous layers. Backpropagation is the most common algorithm for training feedforward neural networks. The steps are to calculate the error between the network's output (prediction) and the actual target output (from the dataset), and then to adjust the weights of the connections between neurons in order to ultimately minimize this error.

A simple feedforward neural network consists of three main components. The *input layer* receives the data, which can take the form of vectors or matrices representing features or observations. The *hidden layers* process the input and automatically extract relevant and meaningful features for the data by minimizing a cost function. Finally, the *output layer* produces task-specific outputs, such as classifications or predictions.

Representation of a DNN: Mathematically, a neuron is represented as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right),$$

where x_i are the inputs received from the previous layer, w_i are the weights that determine the importance of each input, b is a bias term that extends the linear transformation to an affine transformation, f is the activation function introducing non-linearity to capture complex patterns, and y is the neuron's output. An illustration of a neuron can be seen in Figure 3.1.

A collection of neurons on the same level constitutes a layer in a neural network, as shown in Figure 3.2. The output of a layer can be represented in matrix form as

$$h^{(l)} = f\left(W^{(l-1)} \cdot h^{(l-1)} + b^{(l-1)}\right),$$

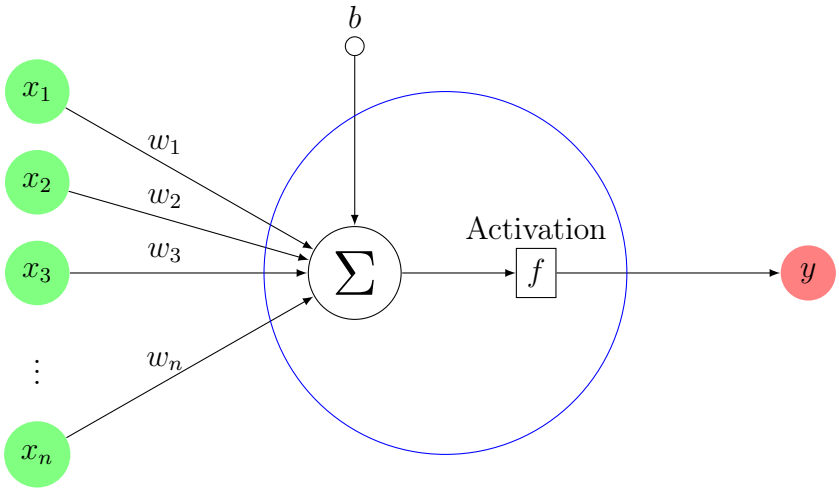


Figure 3.1: A neuron with n inputs x_i, their associated weights w_i, and the summation node Σ. The activation function f processes the weighted sum of the inputs and produces the output y. Inspired by [2].

where $x^{(l-1)}$ is the input vector to layer l (stacked outputs of the previous layer), $W^{(l-1)}$ is the weight matrix for layer l (representing all the weights from layer $l-1$ to layer l), $b^{(l-1)}$ is the bias vector (stacked bias weights from layer $l-1$ to layer l), and $y^{(l)}$ is the output vector of the layer (stacked outputs of the neurons of layer l). The network can have any number of hidden layers allowing it to model more or less complex relationships in the data.

Figure 3.3 visually represents a neural network, showing how the input layer, hidden layers, and output layer work together, with each node in the network representing a neuron. The information flows from layer to layer, with weights determining the strength of connections between neurons.

Types of Activation Functions: The weighted sum of inputs for each neuron is passed through an activation function to inject non-linearity, enabling the network to capture complex patterns in data. Below are some of the key activation functions used in neural networks:

1. **Linear Activation Function:**
$$f(x) = x$$
A neural network with linear activation functions would result in a purely linear model.

2. **Rectified Linear Unit (ReLU):**
$$f(x) = \max(0, x)$$
ReLU outputs zero for negative inputs and a linear response for positives. Intuitively, the multiple layers using a ReLU activation function aim to separate the data space into subspaces defined by polyhedra and then solve the task (e.g. classification).

3. **Leaky ReLU:**
$$f(x) = \max(\alpha x, x)$$
This is a variant of ReLU that ensures neurons continue to learn even when receiving negative inputs, using a small positive constant α.

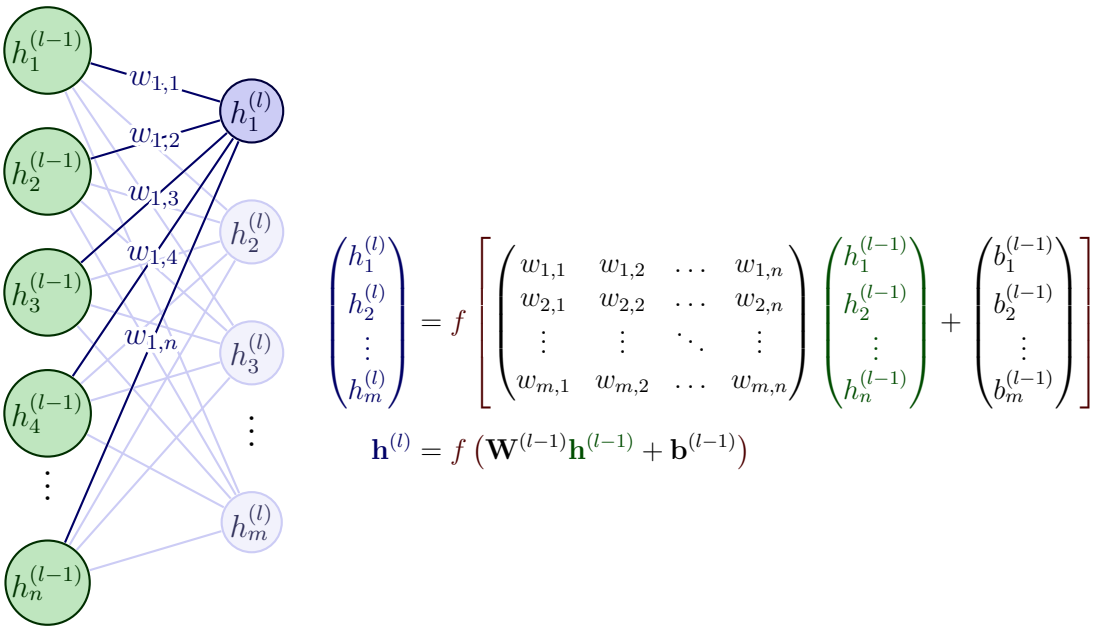


Figure 3.2: A representation of the neural network layer computation. The output of layer l, denoted as $h^{(l)}$, is computed by applying an activation function f to the weighted sum of the previous layer's output $h^{(l-1)}$ through the weight matrix $W^{(l-1)}$, plus a bias term $b^{(l-1)}$. Inspired by [3]

4. **Sigmoid Function:**

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

   This function compresses the real input into the range $(0, 1)$, which can help interpret the output as a probability.

5. **Hyperbolic Tangent (tanh) Function:**

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

   The tanh function maps inputs into the range $(-1, 1)$.

6. **Softmax Function:**

$$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

   This function transforms an n-dimensional input vector into a probability distribution and is useful in multi-class classification scenarios.

Figure 3.4 shows a visual representation of the activation functions mentioned.

Neural Network Training and Learning The goal of a deep learning model for a supervised learning task is to find its parameters θ that minimize the empirical risk:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} L\big(f(x_i; \theta), y_i\big),$$

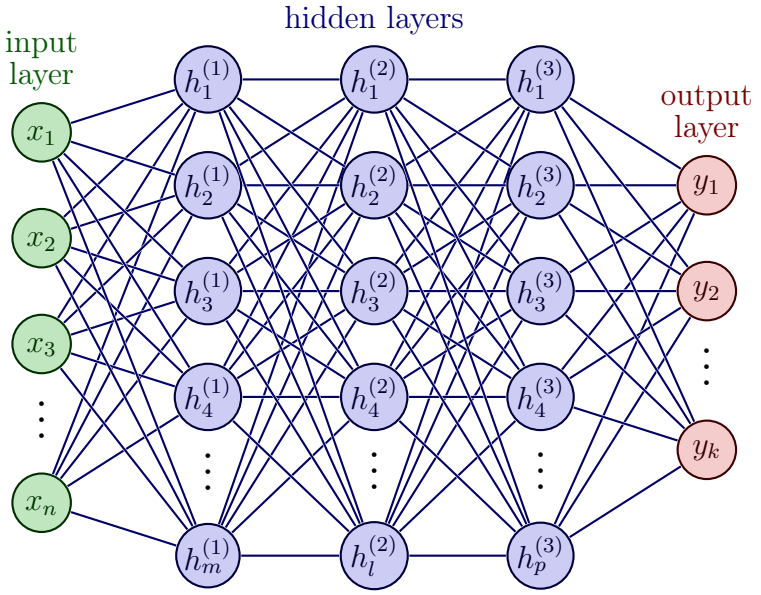


Figure 3.3: Visualization of a Feedforward Neural Network with Input, Hidden, and Output Layers. Each node represents a neuron, and the connections illustrate the flow of information. Source: [3].

where $f(x_i;\theta)$ is the output of the neural network for input x_i, and $L(\cdot,\cdot)$ is a loss function which measures the discrepancy between the predictions $\hat{y}_i$ and the ground truth y_i. Some common loss functions include the Mean Squared Error (MSE):

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

the Binary Cross-Entropy (BCE):

$$L_{\text{BCE}} = -\frac{1}{n}\sum_{i=1}^{n}\Big[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\Big],$$

and the Cross-Entropy (CE) loss for multi-class classification:

$$L_{\text{CE}} = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}).$$

The training of a network is usually based on gradient optimization methods. For instance, a gradient descent update is:

$$x(t+1) = x(t) - \alpha\nabla f\big(x(t)\big),$$

where $\alpha > 0$ is the learning rate. In practice, stochastic gradient descent (SGD) is used to update the parameters based on mini-batches of training data. The efficient computation of the gradient $\nabla_\theta J(\theta)$ in multi-layer networks is achieved using the backpropagation algorithm, which usess the chain rule. For a neuron with output $y = f(z)$ where $z = \sum_i w_i x_i + b$, the chain rule gives

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i}.$$

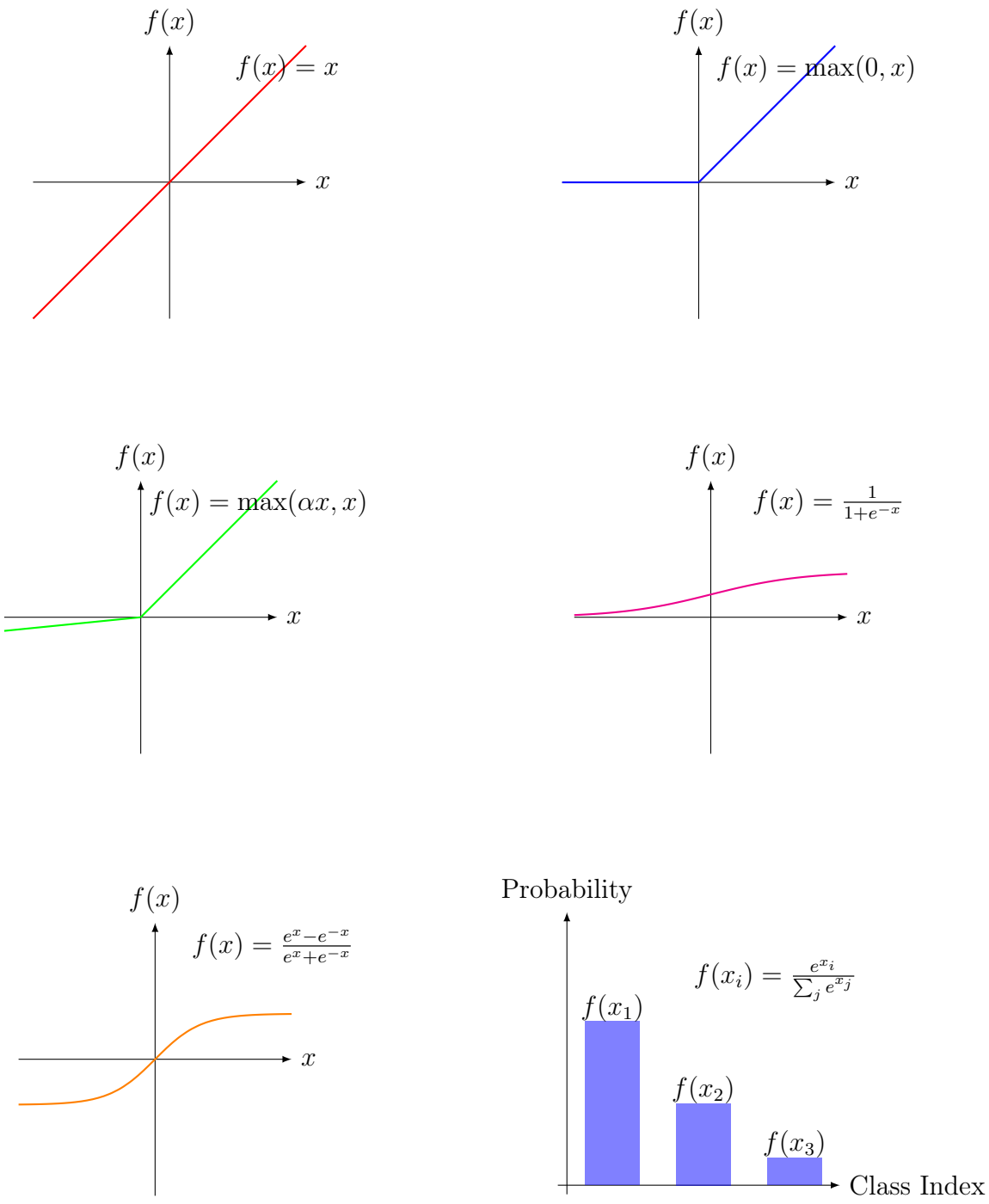


Figure 3.4: Linear Activation Function (Top Left), ReLU Activation Function (Top Right), Leaky ReLU Activation Function (Middle Left), Sigmoid Activation Function (Middle Right), Tanh Activation Function (Bottom Left), and a bar chart representation of Softmax Activation Function (Bottom Right).

The backpropagation procedure involves the forward pass to compute the network output for each data sample, the loss computation given an output and the corresponding ground truth, and the backward pass where gradients are propagated from the output back through each layer, to update the network parameters, as shown in Figure 3.5.

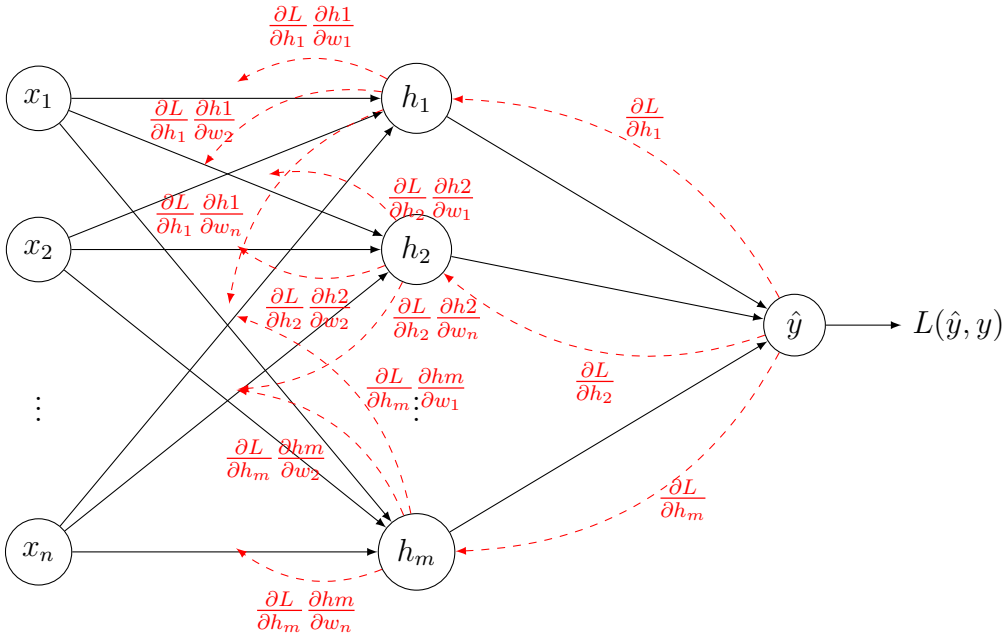


Figure 3.5: Neural Network Architecture with Forward and Backward Pass

3.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs), introduced by [11], are specialized neural networks designed to recognize spatial dependencies within data. Especially for image processing tasks, CNNs were designed to automatically take into account pixels in the neighborhood of each pixel through convolution, thus handling spatial structures. They adjust the weights of filters (kernels) during training to learn important local patterns in the data, such as edges, textures, and shapes, in a process named feature extraction.

Convolution: The convolution operation is used for feature extraction by applying a filter (kernel) over the input tensor. Suppose we have an input image $\mathbf{I}$ of dimensions $m_1 \times m_2 \times m_c$, where m_1 and m_2 are the height and width, and m_c is the number of channels. We apply a filter $\mathbf{K}$ of dimensions $n_1 \times n_2 \times n_c$, where $n_c = m_c$. The convolution operation is given by:

$$F[i,j] = (I * K)_{[i,j]} = \sum_{x=1}^{n_1} \sum_{y=1}^{n_2} \sum_{z=1}^{n_c} K[x,y,z] \cdot I[i+x-1, j+y-1, z],$$

where $F[i,j]$ is the (i,j)-th entry of the feature map, and $K[x,y,z]$ and $I[i+x-1, j+y-1, z]$ represent the kernel and input image at specific positions, respectively. After applying the convolution, an activation function is used on the result to introduce non-linearity. For example, applying the ReLU activation function:

$$F[i,j] \leftarrow \max(0, F[i,j]).$$

Padding: To maintain the spatial dimensions or prevent the convolution from focusing only on the central regions, padding is applied. For example, with zero-padding, a row and column of zeros are added on all sides of the input tensor, allowing the kernel to scan the borders of the image.

Pooling: Pooling is a downsampling operation that reduces the spatial size of the feature maps, preserving dominant features. Max pooling selects the maximum value within a local region. For example, for a 2×2 patch:

$$y[i,j] = \max_{x=1}^{2} \max_{y=1}^{2} F[i+x-1, j+y-1].$$

Pooling makes the network computationally efficient and reduces overfitting by aggregating the features in various regions.

Fully Connected Layer: After several convolution and pooling layers, the output feature maps are flattened into a single vector, which is passed through fully connected layers. This layer combines the output into a new vector, representing the final classification or regression. The fully connected layer is computed as:

$$y = f(W \cdot \text{vec}(F) + b),$$

where W is the weight matrix, $\text{vec}(F)$ is the flattened feature map vector, b is the bias, and f is the activation function. For the hidden layers, a common choice for the activation functions is the ReLU, while for the output layer, if classification is the task, the softmax function is commonly used:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}},$$

where z_i is the output for class i, and $\sigma(z_i)$ gives the probability distribution over all possible classes.

An example of a convolutional neural network architecture, can be seen in Figure 3.6.

3.1.3 Recurrent Neural Networks (RNNs)

RNNs, introduced at [12, 13], are designed for sequential data, such as time series or natural language, where previous inputs influence future outputs. The fundamental concept of RNNs is that their hidden state is recursively updated. Specifically, at each time step t, the hidden state h_t is updated based on the current input x_t and the previous hidden state h_{t-1}:

$$h_t = f(W_h \cdot h_{t-1} + W_x \cdot x_t + b_h),$$

where W_h and W_x are weight matrices, b_h is a bias vector, and f is a non-linear activation function, often tanh or ReLU. The output of the RNN at each time step, denoted y_t, is computed as:

$$y_t = g(W_y \cdot h_t + b_y),$$

where W_y is the output weight matrix, b_y is the bias term, and g is an activation function tailored to the task. For classification, g is typically the softmax function, whereas for regression tasks, a linear activation is used. Figure 3.7 illustrates the architecture of a typical RNN.

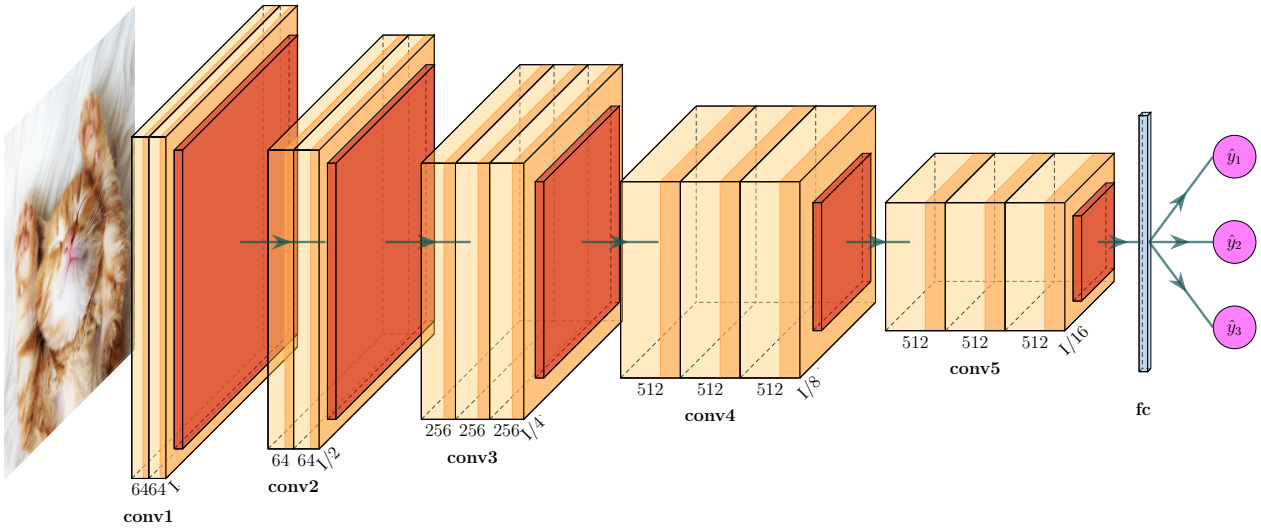


Figure 3.6: Example architecture of a Convolutional Neural Network. Source: [4]

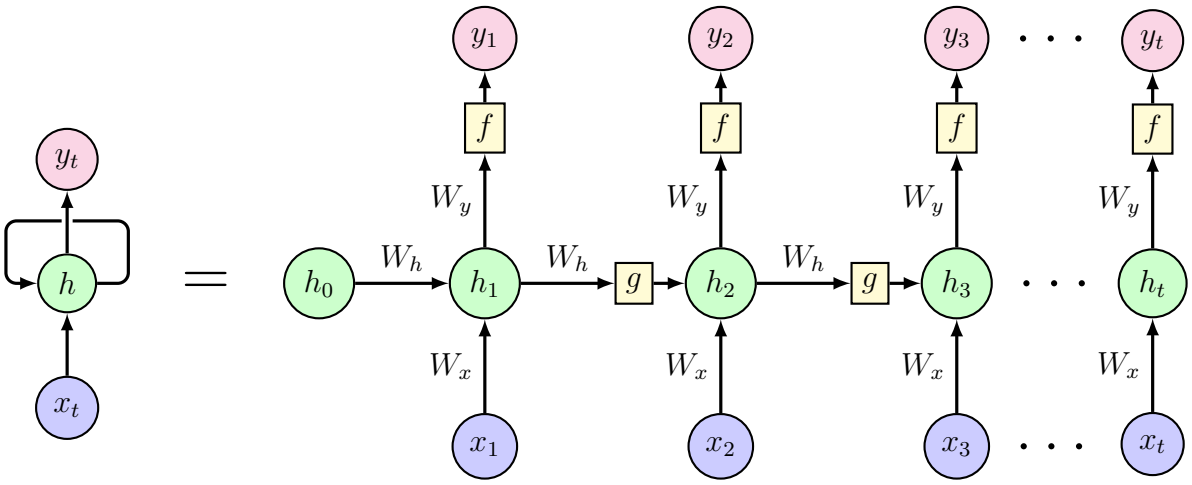


Figure 3.7: Recurrent architecture with nodes representing hidden states, inputs, outputs, and transformations.

Long Short-Term Memory Networks: Long Short-Term Memory (LSTM) networks [14] extend the capabilities of RNNs by addressing the vanishing[1] and exploding[2] gradient problems. LSTMs use gating mechanisms to regulate the flow of information through the network. The output y_t of an LSTM at each time step is computed in a similar manner to RNNs:

$$y_t = g(W_y \cdot h_t + b_y),$$

[1]The *vanishing gradient problem* occurs when gradients become extremely small during backpropagation through many time steps (due to the repeated multiplication of small derivatives), making it difficult for the network to learn long-term dependencies.

[2]The *exploding gradient problem* occurs when gradients become excessively large (due to the repeated multiplication of small derivatives), leading to unstable weight updates.

where g is often a softmax or linear activation (depending on the task) and various gates control the flow of information:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) && \text{(forget gate)}, \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) && \text{(input gate)}, \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) && \text{(cell state update)}, \\ C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t && \text{(new cell state)}, \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) && \text{(output gate)}, \\ h_t &= o_t \odot \tanh(C_t) && \text{(new hidden state)}, \end{aligned}$$

where $\odot$ denotes element-wise multiplication.

The *forget gate*, f_t, determines how much of the previous cell state (C_{t-1}) should be retained. If f_t is close to 1, the corresponding information is preserved. If f_t is close to 0, the information is discarded. The *input gate*, i_t, decides how much of the new candidate information ($\tilde{C}_t$) is added to the cell state. This controls how new information flows into the LSTM. The *candidate cell state*, $\tilde{C}_t$, represents new information generated based on the current input (x_t) and the previous hidden state (h_{t-1}). The tanh activation function ensures these values are within the range $[-1, 1]$. The *new cell state*, C_t, is computed by combining the old cell state, scaled by the forget gate, and the candidate cell state, scaled by the input gate. This mechanism allows the LSTM to selectively retain past information and incorporate new observations. The *output gate*, o_t, controls how much of the updated cell state contributes to the new hidden state. The *hidden state*, h_t, combines the cell state and output gate. The cell state is passed through a tanh activation function, and the output gate determines what part of this processed state is output as the hidden state. The final output, y_t, depends on the hidden state and the activation function (g), which may be a softmax or linear activation, depending on the task. A typical LSTM cell is illustrated in Figure 3.8, and a LSTM network consists of multiple LSTM cells connected serially and possible has more than one layer. A LSTM variant is the Bi-LSTM, which consists of two LSTM networks that parse information in opposite ways, in order to capture more complex dependencies.

Gated Recurrent Unit: Another popular variant of RNNs is the Gated Recurrent Unit (GRU), introduced by [20], which simplifies the architecture of LSTMs by combining certain gates. The update and reset gates of GRUs, denoted z_t and r_t respectively, are computed as:

$$\begin{aligned} z_t &= \sigma(W_z \cdot [h_{t-1}, x_t] + b_z) && \text{(update gate)}, \\ r_t &= \sigma(W_r \cdot [h_{t-1}, x_t] + b_r) && \text{(reset gate)}, \\ \tilde{h}_t &= \tanh(W \cdot [r_t \odot h_{t-1}, x_t] + b) && \text{(candidate hidden state)}, \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t && \text{(new hidden state)}, \end{aligned}$$

where W_z, W_r are the weight matrices, b_z, b_r are bias vectors, and σ is the sigmoid activation function. The candidate hidden state $\tilde{h}_t$ is then calculated as:

$$\tilde{h}_t = \tanh(W \cdot [r_t \odot h_{t-1}, x_t] + b),$$

where $\odot$ denotes element-wise multiplication. The output of the GRU is derived similarly to that of RNNs and LSTMs, using the hidden state h_t and the output weight matrix W_y:

$$y_t = g(W_y \cdot h_t + b_y).$$

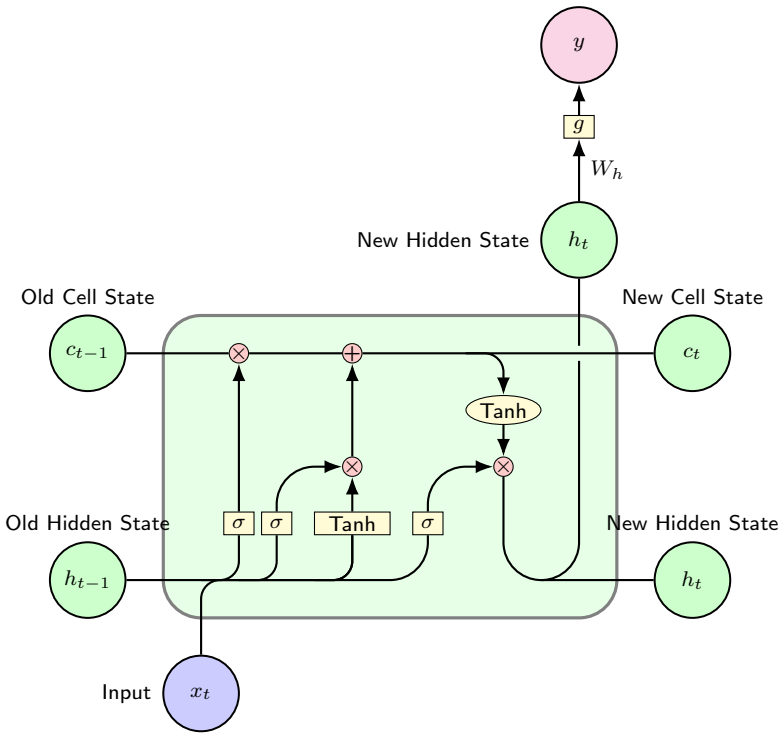


Figure 3.8: Representation of a typical LSTM cell structure. Inspired by [5].

The *update gate*, z_t, determines the extent to which the previous hidden state (h_{t-1}) is retained. A value close to 1 means most of the past information is kept, while a value close to 0 means the new hidden state depends more on the current input. The *reset gate*, r_t, controls how much of the past hidden state (h_{t-1}) contributes to the candidate hidden state ($\tilde{h}_t$). When r_t is close to 0, the network "forgets" most of the previous state and focuses on the current input. The *candidate hidden state*, $\tilde{h}_t$, is computed using the reset gate, past hidden state, and current input. It represents the potential new state for the GRU. The *new hidden state*, h_t, is a weighted combination of the previous hidden state and the candidate hidden state, controlled by the update gate. Specifically, $(1 - z_t) \odot h_{t-1}$ represents the part of the old state that is retained, while $z_t \odot \tilde{h}_t$ integrates the newly computed information. A typical GRU cell is illustrated in Figure 3.9.

3.1.4 Autoencoders

Autoencoders [15] are unsupervised learning models that aim to learn a meaningful representation of the input data, from which they can reconstruct this input. They consist of two components: the encoder and the decoder. The encoder maps the input data x into a compressed latent representation z, capturing its most essential features. This transformation is expressed as:

$$z = f(W_e \cdot x + b_e),$$

where W_e and b_e are learnable parameters, and f is a nonlinear function. The decoder reconstructs the input from z using:

$$\hat{x} = g(W_d \cdot z + b_d),$$

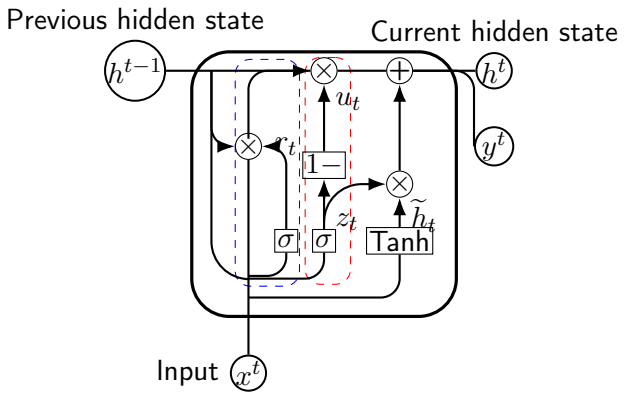


Figure 3.9: Typical Gated Recurrent Unit (GRU) architecture. Inputs h^{t-1} and x^t denote the previous hidden state and the current input to the GRU respectively. Outputs h^t and y^t denote the current hidden state and a processed output (such as softmax of h^t) respectively. The red dashed box denotes the Update Gate and the blue dashed box denotes the Reset Gate. The Reset Gate output is denoted by r_t, the Update Gate output is given by z_t, while the candidate hidden state is $\widetilde{h}_t$.

where g is another nonlinear function that transforms the latent representation back into the original input space.

The autoencoder is trained to minimize the reconstruction loss, which is the difference between the original input x and its reconstruction $\hat{x}$. The loss function is usually the mean squared error (MSE):

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \|x_i - \hat{x}_i\|^2,$$

where n is the number of samples in the dataset. By minimizing this loss, the autoencoder learns to encode the input into a lower-dimensional latent space that retains the most useful information for reconstruction.

Variational Autoencoders: Autoencoders have been extended to solve specific problems. For instance, variational autoencoders (VAEs) [21] are the probabilistic version of Autoencoders, where the encoder learns to map the input to parameters of a prior probability distribution, enabling the decoder to generate new data samples by sampling the distribution and mapping the samples back to the original space. We will discuss VAEs extensively later, as their concepts are fundamental for *diffusion models*, which is a central component of this thesis. Denoising autoencoders [22] are trained to reconstruct the original input from a corrupted version, making them robust to noise and capable of feature extraction in noisy environments. Sparse autoencoders enforce sparsity in the latent representation by using regularization terms. Autoencoders are also used in anomaly detection, where reconstruction errors can disclose deviations from normal patterns. Figure 3.10 visualizes the typical architecture of an Autoencoder.

Transformers: Transformers, introduced in [16], revolutionized sequence modeling by using attention mechanisms. Transformers can process entire sequences, improving efficiency and scalability. A self-attention mechanism computes relationships between all positions in the input sequence. Given an input sequence $X = [x_1, x_2, \ldots, x_n]$, the self-attention generates representations, which account for the contexts, by weighing the relevance of other elements to each position.

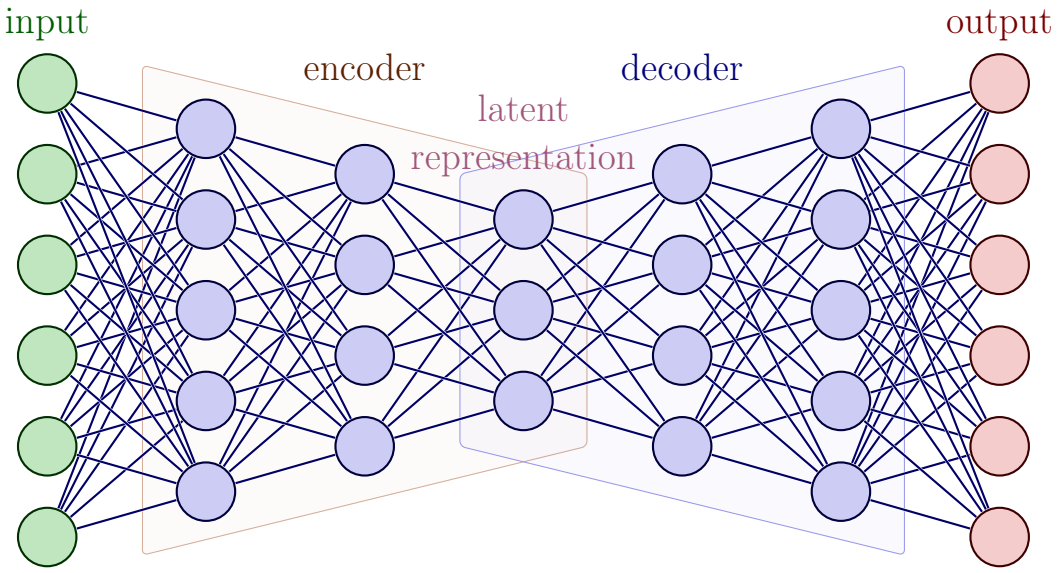


Figure 3.10: Structure of an autoencoder neural network. The left section represents the encoder, and the right section represents the decoder. Source: [3].

The self-attention mechanism operates through queries (Q), keys (K), and values (V), which are matrices derived from the input. The output is computed as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

where d_k is the key dimensionality, used for scaling to stabilize training. Multi-head attention extends this mechanism by projecting inputs into multiple subspaces, allowing the model to focus on diverse features. The architecture also utilizes positional encodings to account for the sequential nature of the data.

3.2 Time Series

3.2.1 Definitions and Practices

Time series data is a sequence of data points measured at successive points in time. A main characteristic is its temporal ordering, where each observation can depend on the previous ones. Other characteristics are *Trend*, which refers to a long-term increase or decrease in the data, *Seasonality*, which indicates a repeating pattern over a period, and *Stationarity*, which is a property where the statistical characteristics, such as mean and variance, remain constant over time. Furthermore, time series can be classified as *univariate*, where only a single variable is observed over time, or *multivariate*, when each point consists of multiple time-dependent variables.

Time Series Data Preprocessing Data preprocessing is an important step in time series analysis and forecasting, like in every task in Data Science. It prepares the data for modeling and enables algorithms to learn patterns and relationships more effectively. Time series data often contains noise, outliers, and missing values, which can lead to issues such as biased estimates, decreased predictive accuracy, loss of temporal context, misleading trends, model instability, and difficulty in interpretation and modeling in general. Thus, cleaning and transforming the data are essential steps to improve a model's performance.

The first step is *Data Cleaning* and handles issues like missing data, outliers, and noise, which make time series analysis hard. They can cause biased model estimates, reduce the predictive ability of models, and disrupt temporal dependencies. Some techniques for addressing these problems are missing values imputation, outliers detection and correction, and smoothing noisy data to preserve only the meaningful patterns.

Scaling techniques such as min-max or standard scaling help to standardize data. They are particularly useful for algorithms that are sensitive to input scales, such as neural networks. These techniques mitigate the bias caused by scale differences, resulting in more robust models.

Many time series models assume stationarity, where statistical properties remain constant over time. *Stationarization* is the process that removes trends to make the series stationary through transformations, such as differencing: $y'_t = y_t - y_{t-1}$, where y'_t is the differenced series, representing the change between consecutive time points.

Feature engineering is another data preprocessing technique that enhances model performance by creating additional variables, from raw data, that capture essential patterns in the time series. **Lag features**, such as y_{t-1}, y_{t-2}, etc, capture temporal dependencies. *Rolling statistics*, like rolling means and standard deviations, smooth short-term fluctuations to highlight trends. *Time-based features*, such as the day of the week or month, make it easier for models to locate temporal relationships. *Spectral features*, like applying Fast Fourier Transform (FFT) on windows of the time series, reveal frequency components of the data, which is particularly useful for identifying periodic patterns and cyclical trends.

Decomposition breaks a time series into three components: *trend*, *seasonality*, and *residuals*. The trend represents the long-term progression of the time series, that is the overall direction that the data is moving in. The seasonality component represents repeating patterns (cycles) in the data that occur at specific intervals (for example daily, weekly, monthly, or yearly). These patterns can be caused by factors such as weather seasons, holidays, work related cycles, etc. Finally, the residual (or noise) component captures the irregular variations in the data, that remain after removing the trend and seasonality components. These variations can be

considered unpredictable and represent the unexplained part of the time series. Mathematically, decomposition is defined as:

$$y_t = Trend_t + Seasonality_t + Residual_t$$

3.2.2 Classical Models for Time Series Prediction

Before discussing modern, more complex models for time series analysis, we first consider some foundational classical models that are simple, yet widely used in predictive tasks.

Beginning with the *Autoregressive* (AR) model, it is a time series model that predicts the future values of the series based on the past values. Mathematically, it is defined as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t,$$

where ϵ_t represents white noise. The parameters $\phi_1, \phi_2, \ldots, \phi_p$ are the coefficients of the lagged terms, and c is a constant. AR models are effective at capturing linear dependencies in time series data, as the prediction is an affine transformation of the previous values, and are suitable for applications in finance. The parameters of the model are estimated using the method of least squares. Given a training dataset, the goal is to minimize the sum of squared residuals:

$$\text{Residual} = y_t - (c + \phi_1 y_{t-1} + \cdots + \phi_p y_{t-p}) .$$

The *Moving Average* (MA) model, in contrast to the AR model, incorporates past forecast errors to model the current value of the time series. The model is given by:

$$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q},$$

where ϵ_t represents white noise, and $\theta_1, \theta_2, \ldots, \theta_q$ are coefficients that measure the impact of past error terms on the current value y_t. This way, it smooths out short-term variations in the data, and emphasizes on longer-term trends and patterns. MA models are applied in signal processing and economics. Training MA models is done by estimating the coefficients $\theta_1, \theta_2, \ldots, \theta_q$ by maximizing the likelihood of the observed data under the assumption that the residuals are Gaussian white noise using optimization techniques like the Expectation-Maximization (EM) algorithm or gradient-based methods.

The *Autoregressive Integrated Moving Average* (ARIMA) model extends the AR and MA models to handle non-stationary data, through the "Integrated" part. Mathematically we have:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \cdots + \theta_q \epsilon_{t-q},$$

where ϵ_t is white noise, ϕ_i are the autoregressive coefficients, and θ_i are the moving average coefficients. To address non-stationarity, ARIMA applies differencing to the data, transforming it into a stationary form, as discussed earlier. In essence, ARIMA models combine the autoregressive component (AR), which captures dependencies on past values, with the moving average component (MA), which captures dependencies on past forecast errors, and uses the integration component (I) to make the data stationary so the former components operate efficiently. These models are widely used for forecasting in various fields such as finance and environmental sciences. Training ARIMA models involves estimating the AR (ϕ) and MA (θ) coefficients and the differencing order (d). The first two parts are estimated through maximum likelihood estimation (MLE), and differencing, along with statistical tests like the Augmented Dickey-Fuller test, are applied iteratively to confirm stationarity.

The *Seasonal ARIMA* (SARIMA) model extends the ARIMA model by incorporating seasonal parts. Mathematically it is formulated as:

$$y_t = \text{ARIMA}(p, d, q) \times \text{Seasonality}(P, D, Q, s).$$

Here, p, d, and q refer to the autoregressive order, the degree of differencing, and the moving average order, respectively. The seasonal components are denoted as P, D, Q, and s, where P is the seasonal autoregressive order, D the seasonal differencing order, Q the seasonal moving average order, and s the length of the seasonal cycle. The ARIMA parameters are p, which defines the number of lag observations the model considers for prediction, d, which is number of differencing operations in order to make the series stationary, and q, which determines the size of the moving average window, which captures dependencies on past forecast errors. The seasonal components enable the model to account for patterns that repeat over a specific cycle s. For example, if data exhibits daily seasonality and the time stamps have a period of 1 hour, s would be set to 24. The training process for SARIMA models is similar to that of ARIMA, with additional seasonal parameters P, D, Q, s to estimate. Seasonal differencing (D) is applied first to remove seasonal trends, and then standard differencing (d) follows. SARIMA models are effective for time series with strong seasonal patterns.

3.2.3 Deep Learning Time Series Models

After examining classical models for time series, we will focus on deep learning methods. Unlike the traditional models, deep learning provides more flexibility and adaptability in capturing complex temporal patterns, non-linear dependencies and features due to the models' inherent complexities. The following models were thoroughly examined in Section 3.1, so we will briefly discuss them in the context of time series analysis and forecasting.

Although typically used for spatial data, *Convolutional Neural Networks* (CNNs) [11] can effectively model temporal dependencies in time series data. In this context, the convolution operation extracts local patterns from a time series, such as short-term trends or seasonal patterns. For example, for a 1D time series $\mathbf{x}_t$, the convolutional layer computes:

$$z_t = \sigma\left(\sum_{k=1}^{K} w_k \cdot x_{t+k-1} + b\right),$$

where w_k are the filter weights, K is the kernel size, and σ is an activation function. By stacking multiple convolutional layers, CNNs can capture increasingly complex patterns over multiple time scales. They are useful for tasks such as multivariate time series forecasting and anomaly detection.

Recurrent Neural Networks (RNNs) [12,13] are designed to process sequential data by maintaining a hidden state $\mathbf{h}_t$ that evolves over time based on the current input $\mathbf{x}_t$ and the previous state:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t).$$

Traditional RNNs, however, have a hard time modeling long-range dependencies due to the vanishing gradient problem. This problem is caused by the repeated multiplication of small gradients during backpropagation, leading to very small updates, and as a result, the network is inefficient in learning long-term dependencies. To address this, Long Short-Term Memory (LSTM) networks [14] and Gated Recurrent Units (GRUs) [20] introduce gating mechanisms

that control the influence of past states and new information to the current state, enabling the model to maintain relevant features over extended sequences.

Attention mechanisms [16] revolutionize time series analysis as they allow models to selectively focus on the most relevant parts of the sequence. In self-attention, the relationship between two time steps t and t' is captured through an attention score:

$$\alpha_{t,t'} = \frac{\exp(\mathbf{q}_t^\top \mathbf{k}_{t'})}{\sum_{t'} \exp(\mathbf{q}_t^\top \mathbf{k}_{t'})},$$

where $\mathbf{q}_t$ (query) and $\mathbf{k}_{t'}$ (key) are learned representations. *Transformers* are built upon this concept and they model dependencies across the entire sequence efficiently. This capability makes them ideal for long-term forecasting tasks and other complex temporal applications.

Sequence-to-Sequence (Seq2Seq) architectures are designed for mapping an input sequence directly to an output sequence. These models consist of an encoder, which maps the input sequence into a context vector $\mathbf{c}$, and a decoder, which generates the forecasted sequence from $\mathbf{c}$ (on the initial space). The combination of attention mechanisms and Seq2Seq models is very powerful, as the total model dynamically weighs the importance of different time steps, and learns representations for time series data in a more robust manner.

3.2.4 Methods for Time Series Forecasting

One-Step vs. Multi-Step Forecasting Time series forecasting methods can be classified based on the prediction horizon.

One-step forecasting focuses on predicting the value at the next immediate time step, given the current and past observations. The benefits of this approach are that it is simple, as the model is trained to predict only one step ahead, and it generally provides more accurate predictions because the prediction horizon is very short, so the model does not require a too complex structure to achieve good results. However, the major drawback of one-step forecasting is its limited scope. The method only provides a single future point, which is limiting for many applications, since longer-term forecasts are usually required (for example in tasks like weather prediction and demand planning). Furthermore, repeatedly applying one-step forecasting to generate multi-step predictions is neither computationally efficient, due to multiple inference processes, nor accurate, because of error accumulation.

On the other hand, *multi-step forecasting* aims to predict a sequence of future values. There are two primary approaches to achieving this: recursive forecasting and direct forecasting. *Recursive forecasting*, also known as iterative method, involves using the model to recursively extend the predictions for the desired horizon, by predicting the next step $t+1$, then feeding this prediction back into the model to predict $t+2$, and so on. As discussed before, this method is consistent, as same model is applied for each prediction, however, recursive forecasting is prone to error propagation, as each successive prediction depends on the previously predicted values. This leads to rapid degradation in accuracy and uncertainty, as the prediction horizon increases. In contrast, *direct forecasting* involves training a model to predict multiple future time steps simultaneously. Instead of generating one prediction at a time, the model outputs the entire forecast horizon $(y_{t+1}, y_{t+2}, \ldots, y_{t+h})$ in a single step. By predicting all future steps directly, this method avoids the issue of error propagation, which is inherent in recursive approaches. However, the complexity of training a direct model is significantly higher, as the model must learn patterns for multiple time steps simultaneously. This also increases computational costs.

Furthermore, a model trained for a specific horizon may not generalize well for other prediction horizons, and in that case, it requires retraining.

Probabilistic Forecasting Contrary to single point estimation, probabilistic forecasting predicts a range of possible outcomes, along with probabilities or confidence intervals. For a given future time step $t + h$, the model outputs a probability distribution $P(y_{t+h} \mid \mathbf{x}_t)$, where $\mathbf{x}_t$ represents the past observed data. The main advantage of probabilistic forecasting is its ability to quantify uncertainty. This is crucial for applications where the environments are uncertain and decisions must account for risk, such as financial markets or energy grid management. For example, instead of predicting energy demand, probabilistic models provide confidence intervals on ranges of possible future outcomes, allowing planners to prepare for best-case, worst-case and average-case scenarios. Additionally, probabilistic forecasting is generally more robust in volatile environments. The significant disadvantage of probabilistic models is that they are often more complex to train compared to point estimating models, as they deal with distributions instead of single values, leading to higher computational costs.

Evaluation Metrics for Time Series Forecasting Evaluating time series forecasting models requires metrics that assess the accuracy and reliability of predictions.

The *mean absolute error* (MAE) measures the average absolute difference between predicted values $\hat{y}_t$ and actual values y_t:

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|.$$

MAE is simple to compute and interpret and is less sensitive to outliers compared to squared-error metrics.

The *mean absolute percentage error* (MAPE) expresses the forecast error as a percentage of the actual values:

$$\text{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n}\frac{|y_t - \hat{y}_t|}{|y_t|}.$$

MAPE is scale-independent, but has limitations when actual values y_t are close to zero, as the percentage error can become very large or undefined.

The *mean squared error* (MSE) calculates the average of squared differences:

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2.$$

MSE is particularly useful when models are needed to avoid large deviations, but its high sensitivity to outliers can sometimes make model training inefficient.

The *root mean squared error* (RMSE) is the square root of the MSE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}.$$

RMSE has the same advantages and drawbacks as MSE but is in the same units as the predicted variable, which is more interpretable.

The *symmetric mean absolute percentage error* (SMAPE) expresses forecast accuracy as a percentage:

$$\text{SMAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{(|y_t| + |\hat{y}_t|)/2}.$$

It is symmetric, penalizing over- and under-predictions equally. However, it may return high values when actual values y_t are near zero.

The *mean absolute scaled error* (MASE) is a scale-independent metric that compares forecast errors to a naive forecasting model:

$$\text{MASE} = \frac{1}{n} \frac{\sum_{t=1}^{n} |y_t - \hat{y}_t|}{\frac{1}{n-1} \sum_{t=2}^{n} |y_t - y_{t-1}|}.$$

For probabilistic models, the *Continuous Ranked Probability Score* (CRPS) [45] is a standard metric. CRPS evaluates the accuracy of predicted probability distributions by comparing them to observed values:

$$\text{CRPS}(F, y) = \int_{-\infty}^{\infty} \big(F(x) - \mathbb{1}(x \geq y)\big)^2 dx$$

where $F(x)$ is the cumulative distribution function (CDF) of the predicted distribution and y is the observed value. CRPS generalizes the MAE to probabilistic forecasts, providing a single score that accounts for both the accuracy and certainty of the predicted distribution. This makes CRPS an essential metric for evaluating probabilistic forecasting models, as it measures how well the predicted distribution aligns with the observed outcomes. The CRPS can be understood as the probabilistic analog of the Mean Absolute Error. While MAE assesses the absolute difference between a point prediction $\hat{y}$ and the true value y, CRPS evaluates the entire forecasted distribution against the observed outcome. It captures two aspects of a good probabilistic forecast: The forecast assigning high probability to values close to ground truth y and the forecast's spread reflecting the actual uncertainty in the prediction.

To understand CRPS intuitively, we have $F(x)$ representing the forecasted probabilities up to a certain value x and $\mathbb{1}(x \geq y)$ is the "truth" expressed as a step function. The difference $F(x) - \mathbb{1}(x \geq y)$ quantifies the error at each x, where if $F(x)$ overestimates or underestimates the probability near the true value y, the error is larger. Squaring and integrating this error over all possible values of x yields the CRPS. A smaller score indicates better alignment between the forecasted distribution and the observed value.

For example, let us consider a forecasted normal distribution $\mathcal{N}(\mu, \sigma^2)$. If the actual value y is close to the mean μ, and the standard deviation σ appropriately captures the uncertainty, the CRPS will be low. Else, if y lies far from μ, or if σ is too large or too small (meaning poor calibration), the CRPS increases. In Figure 3.11, the left plot shows this normal distribution with the forecast mean at μ and the true value y marked in red. If the actual value y is close to the mean μ, and the standard deviation σ appropriately captures the uncertainty, the CRPS will be low. This indicates that the forecasted distribution accurately represents the observed value, as the forecast assigns high probability to values near the ground truth y. However, if y lies far from μ, or if σ is too large or too small, the CRPS increases. The right plot shows the cumulative density function (CDF) $F(x)$ of the forecast. The shaded areas represent the errors CRPS metric. To have a good (low) CRPS metric across the values of the time series, μ needs to lie close to y at each time step, in order for the area to be as low as possible, and the spread σ needs to be appropriate so that the combined error of all predictions is as low as possible.

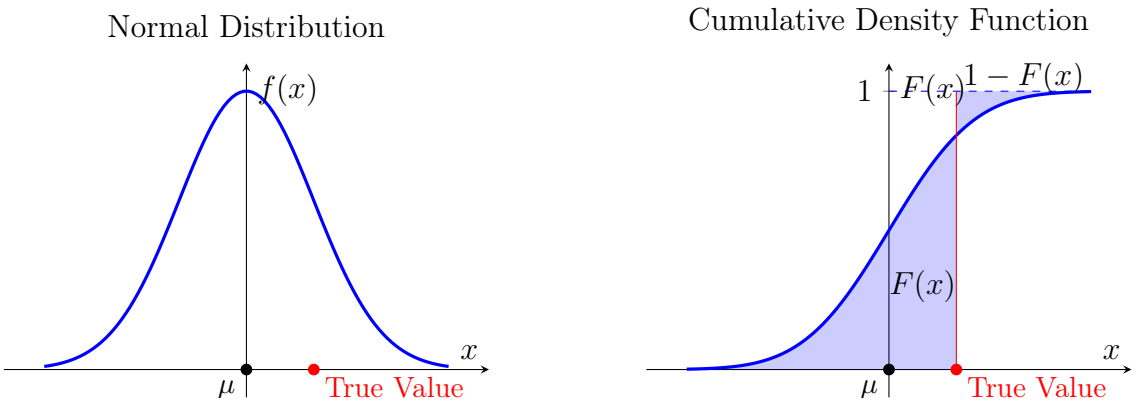


Figure 3.11: Illustration of the Continuous Ranked Probability Score (CRPS). The left plot shows the normal distribution of the forecast, with the true value marked in red. The right plot shows the cumulative density function (CDF) with shaded areas representing the error metric.

3.3 Generative Models for Time Series

Generative models are a class of machine learning models that learn the probability distribution of data. After training, these models can generate new data that are similar to the original data, by sampling the learned probability.

Kevin P. Murphy explains in [17] that a generative model can be described as a joint probability distribution $p(\mathbf{x})$, for $\mathbf{x} \in \mathcal{X}$, where $\mathcal{X}$ is the space of the data distribution. If the model is conditioned on additional information, called covariates $\mathbf{c} \in \mathcal{C}$, it is referred to as a conditional generative model of the form $p(\mathbf{x}|\mathbf{c})$. There are various kinds of generative models that will be analyzed. Common training methods are maximum likelihood estimation (MLE), variational inference and adversarial learning. Once trained, these models can sample from the learned distribution and generate data that look similar to the real data. In the following sections we will discuss the most important generative models and their contributions.

3.3.1 Gaussian Mixture Models (GMM)

We begin with *Gaussian Mixture Models* (GMMs) [18], which are of the most classical and fundamental methods that model data distributions, due to their flexibility and interpretability. GMMs assume that data is generated by sampling from a mixture of several Gaussian distributions. Mathematically, the likelihood of a data point $\mathbf{x}$ under a GMM with K components is represented as:

$$p(\mathbf{x}|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k),$$

where π_k represent the mixture weights (with $\sum_{k=1}^{K} \pi_k = 1$), and $\mathcal{N}(\mathbf{x}|\mu_k, \Sigma_k)$ is the Gaussian distribution for the k-th component with mean μ_k and covariance Σ_k. The mixture weights determine how much each Gaussian component contributes to the overall distribution. The model is trained using the Expectation-Maximization (EM) algorithm [19], alternating between estimating the probability that each data point belongs to each Gaussian (E-step) and then updating the model parameters (M-step).

3.3.2 Hidden Markov Models (HMM)

Hidden Markov Models (HMMs) [46] are statistical models used to represent sequential data where the system is assumed to have hidden (unobservable) states. An HMM is defined by the following components: a set of N hidden states $\{s_1, s_2, \ldots, s_N\}$; transition probabilities $A = \{a_{ij}\}$, where $a_{ij} = p(s_{t+1} = j|s_t = i)$ denotes the probability of transitioning from state s_i to s_j; emission probabilities $B = \{b_j(o)\}$, where $b_j(o) = p(o_t|s_t = j)$ is the probability of observing o_t given the hidden state s_t; and initial state probabilities $\pi = \{\pi_i\}$, where $\pi_i = p(s_1 = i)$ represents the probability that the initial state is s_i. The joint probability of a sequence of observations $\mathbf{O} = \{o_1, o_2, \ldots, o_T\}$ and hidden states $\mathbf{S} = \{s_1, s_2, \ldots, s_T\}$ can be expressed as:

$$p(\mathbf{O}, \mathbf{S}) = \pi_{s_1} b_{s_1}(o_1) \prod_{t=2}^{T} a_{s_{t-1}, s_t} b_{s_t}(o_t).$$

The parameters of an HMM can be estimated using the Baum-Welch algorithm [47], a variant of the Expectation-Maximization (EM) algorithm, which iteratively optimizes the likelihood of the observed data.

3.3.3 RNN-based Generative Models

Recurrent Neural Networks (RNNs) (and their variants like LSTM and GRUs) [12, 14, 20] are suitable for time series generation. Given an initial hidden state $\mathbf{h}_0$, the RNN generates a sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$, where at each time step t, the hidden state is updated based on the previous hidden state and the current input:

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t).$$

The next step in the sequence is then sampled conditionally on the current hidden state:

$$\mathbf{x}_{t+1} \sim p(\mathbf{x}_{t+1}|\mathbf{h}_t).$$

RNN-based generative models capture long-term dependencies and account for the sequential nature of data, thus are efficient for tasks like time series forecasting, music generation and language modeling. We mention in a following section how RNNs are combined with Variational Autoencoders to generate time series data.

3.3.4 Variational Autoencoders (VAEs)

Variational Autoencoders (VAEs) [21] use neural networks and variational inference to model complex distributions. VAEs make the assumption that the data $\mathbf{x}$ is generated from latent variables $\mathbf{z}$ via a decoder $p_\theta(\mathbf{x}|\mathbf{z})$ and these latent variables can be sampled from a prior distribution $p(\mathbf{z})$ which typically is a simple distribution like the normal. The goal is to approximate the posterior $p_\theta(\mathbf{z}|\mathbf{x})$, which describes how the latent variables $\mathbf{z}$ are distributed given the observed data $\mathbf{x}$, using a variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$, through a neural network.

One important challenge that variational inference models have (which is also a challenge for diffusion models and will be discussed again later) is that directly computing the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is intractable because applying Bayes' Rule to find the posterior requires computing the marginal likelihood of the data, $p_\theta(\mathbf{x})$. The true posterior describes how likely a latent variable $\mathbf{z}$ is given a data point $\mathbf{x}$ and is expressed through Bayes' rule as:

$$p_\theta(\mathbf{z} \mid \mathbf{x}) = \frac{p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}.$$

And the marginal likelihood of the data $\mathbf{x}$ is the summation over all latent variables $\mathbf{z}$ of the likelihood of the data $\mathbf{x}$ given the latent variable $\mathbf{z}$ weighted by the prior $p(\mathbf{z})$:

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z})\, d\mathbf{z}.$$

The last integral is intractable in practice because it involves integrating over all possible values of the latent variable $\mathbf{z}$, which becomes computationally prohibitive for high-dimensional or complex models. Directly computing it is infeasible for most models, thus VAEs use a variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$ (with tunable parameters ϕ) to approximate the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$. The parameters of $q_\phi(\mathbf{z}|\mathbf{x})$ are optimized to make the approximation as close as possible to the true posterior. VAEs aim to maximize the marginal likelihood $p_\theta(\mathbf{x})$. However, since computing $p_\theta(\mathbf{x})$ is intractable, they instead optimize a lower bound on it, the Evidence Lower Bound (ELBO).

The logarithm of the marginal likelihood is:

$$\log p_\theta(\mathbf{x}) = \log \int p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z})\, d\mathbf{z}.$$

Using the variational distribution $q_\phi(\mathbf{z}|\mathbf{x})$, we can rewrite this as:

$$\log p_\theta(\mathbf{x}) = \log \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\, d\mathbf{z} = \log \int p_\theta(\mathbf{x}|\mathbf{z})p(\mathbf{z})\frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})}\, d\mathbf{z} =$$
$$= \log \int q_\phi(\mathbf{z}|\mathbf{x})\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\, d\mathbf{z} = \log \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$

The logarithm is a concave function, therefore we can use Jensen's inequality:

$$\log p_\theta(\mathbf{x}) = \log \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$

We can rewrite the RHS term as:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{x} \mid \mathbf{z})p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]$$

Where, if we view the last term as an integral, we get:

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right] = \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{p_\theta(\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\, d\mathbf{z} = -D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$$

This term, $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$, represents the Kullback-Leibler (KL) divergence between the variational posterior and the true posterior. The KL divergence measures the "distance" between the variational posterior and the prior. So, the inequality gives us the Evidence Lower Bound (ELBO) denoted $\mathcal{L}$, which we aim to maximize:

$$\log p_\theta(\mathbf{x}) \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) = \mathcal{L}(\theta, \phi; \mathbf{x}).$$

The first term, $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x}|\mathbf{z})\right]$, encourages the model to reconstruct the observed data well by maximizing the likelihood of the data given the latent variables and the second term, $\mathrm{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))$, regularizes the latent space by ensuring that the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ does not deviate too far from the prior distribution $p(\mathbf{z})$.

Maximizing the ELBO is a tractable problem when it is done using the variational approximation $q_\phi(\mathbf{z}|\mathbf{x})$. Furthermore, the efficient training of VAEs relies on the reparameterization trick which allows gradients to propagate through the stochastic latent variables $\mathbf{z}$. Instead of sampling $\mathbf{z}$ directly from $q_\phi(\mathbf{z}|\mathbf{x})$, which would break the computation graph for gradient-based optimization (because the operation is stochastic and non-differentiable with respect to the parameters ϕ), the reparameterization trick rewrites the sampling process as a deterministic transformation of a noise variable. Specifically, for a Gaussian posterior, we sample $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ and express:

$$\mathbf{z} = \mu_\phi(\mathbf{x}) + \sigma_\phi(\mathbf{x}) \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

This trick allows gradients to flow through μ_ϕ and σ_ϕ, enabling the training via backpropagation. thus enabling backpropagation.

3.3.5 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [23] are composed of two neural networks that compete with each other: the generator G and the discriminator D. The generator G takes as input a

random noise vector $\mathbf{z} \sim p(\mathbf{z})$, where $p(\mathbf{z})$ is usually a simple prior distribution like a normal $\mathcal{N}(0, \mathbf{I})$, and generates a sample $G(\mathbf{z})$ which is supposed to look like the real data $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$. The discriminator D takes a sample (real or generated) and returns a probability that the sample is real. The training of a GAN is a two-player minimax game where the generator aims to "fool" the discriminator, while the discriminator tries to correctly classify the real and generated samples. The training objective is:

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \left[\log D(\mathbf{x})\right] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \left[\log \left(1 - D\left(G(\mathbf{z})\right)\right)\right].$$

In this formulation, the discriminator's goal is to maximize the likelihood of correctly classifying real samples $\mathbf{x} \sim p_{\text{data}}$ as real and to minimize the likelihood of classifying generated samples $G(\mathbf{z})$ as real. On the contrary, the generator's goal is to minimize $\log\left(1 - D\left(G\left(\mathbf{z}\right)\right)\right)$, meaning it tries to "fool" the discriminator into classifying the generated samples as real. The desired outcome is to train the generator to create realistic samples that the discriminator cannot distinguish from the real ones.

The important challenge in training is that the behavior of the networks can be unstable and hard to predict. Work [48] examines the equilibria of GANs.

3.3.6 Normalizing Flows

Normalizing Flows [24] is another flexible and powerful method to model complex probability distributions. The idea is to use a series of invertible transformations in order to transform a simple base distribution (like a Gaussian) into the complex data distribution. These models start with a latent variable $\mathbf{z}_0 \sim p(\mathbf{z}_0)$, sampled from a simple distribution and apply a sequence of invertible transformations $f_1, f_2, \ldots, f_K$ to obtain $\mathbf{z}_K = f_K \circ f_{K-1} \circ \cdots \circ f_1(\mathbf{z}_0)$.

This way, normalizing flows allow the exact evaluation of the likelihood. The probability density function of $\mathbf{z}_K$ under the transformed distribution is:

$$p(\mathbf{z}_K) = p(\mathbf{z}_0) \left|\det \frac{\partial f_K^{-1}}{\partial \mathbf{z}_K}\right| \ldots \left|\det \frac{\partial f_1^{-1}}{\partial \mathbf{z}_0}\right| = p(\mathbf{z}_0) \prod_{k=1}^{K} \left|\det \frac{\partial f_k^{-1}}{\partial \mathbf{z}_{k-1}}\right|$$

The Jacobian determinant $\left|\det \frac{\partial f_k^{-1}}{\partial \mathbf{z}_{k-1}}\right|$ shows how the volume of probability mass changes under each transformation. Normalizing flows can efficiently model complex, high-dimensional distributions with exact likelihoods and easy sampling.

3.3.7 Generative Adversarial Networks for Time Series

Generative Adversarial Networks can be extended for time series analysis with models like TimeGAN [25] and TS-GAN [26]. These models incorporate recurrent neural network components in both the generator and discriminator to handle sequential data.

In *TimeGAN*, the generator G generates time series data $\hat{\mathbf{x}}_t$ attempting to mimic the real data $\mathbf{x}_t$, while the discriminator D tries to distinguish between real and synthetic sequences. Additionally, an embedding network maps the data into a latent space while preserving the sequence's temporal relations. The generator, in a similar manner as normal GANs, learns to generate realistic sequences while at the same time the embedding network is responsible to keep the temporal coherence. The loss function of TimeGAN has two components. The Adversarial Loss $\mathcal{L}_{\text{adv}}$, which pushes the generator to produce data similar to the real data, like

standard GANs, and the Supervised Reconstruction Loss $\mathcal{L}_{\text{supervised}}$, which pushes the generator to be accurate in the latent space reconstruction by preserving the temporal dependencies in the generated sequences. The total objective is a weighted sum of the loss terms:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{adv}} + \lambda \mathcal{L}_{\text{supervised}},$$

where the hyperparameter λ controlls the trade-off between the two losses. In a similar manner, TS-GAN use RNNs for both the generator and discriminator to generate and classify sequential data.

3.3.8 Variational Recurrent Autoencoders (VRAE)

Variational Recurrent Autoencoders (VRAEs) [27] are extended VAEs that handle time series data by combining RNNs with variational inference. A VRAE maps (encodes) a sequence $\mathbf{x}_1, \ldots, \mathbf{x}_T$ to a latent space using a recurrent encoder, and then reconstructs the sequence (mapping back to the original space) using a recurrent decoder. The latent variables $\mathbf{z}$ are sampled from a variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ in order to condition the decoder. The VRAE loss function consists of two main components:

$$\mathcal{L}_{\text{VRAE}} = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \left[\log p_\theta(\mathbf{x}|\mathbf{z})\right] - D_{\text{KL}}(q_\phi(\mathbf{z}|\mathbf{x})||p(\mathbf{z})).$$

The first term, reconstruction loss, encourages the model to reconstruct the input sequence and the second term, the KL divergence, regularizes the latent space, ensuring that the variational posterior $q_\phi(\mathbf{z}|\mathbf{x})$ does not deviate too far from the prior $p(\mathbf{z})$, exactly as VAEs do. This way, VRAEs can incorporate uncertainty in time series modeling.

3.4 Diffusion Models

Diffusion Models are a class of latent variable generative models that have gained popularity in recent years for their incredible ability to model complex data distributions. Inspired by non-equilibrium thermodynamics and statistical physics, they define a Markov chain of diffusion steps to slowly add random noise to data and then learn to reverse this process to reconstruct the data. This concept will be thoroughly discussed in the following sections which analyze some of the fundamental works on Diffusion Models.

3.4.1 Deep Unsupervised Learning using Nonequilibrium Thermodynamics

This class of models, introduced in [28], is inspired by principles from statistical physics and nonequilibrium thermodynamics. The core idea is to model the generation process as the inverse of a diffusion process that gradually destroys structure in the data by adding noise.

As illustrated in Figure 3.12, the forward diffusion is like the dispersal of color in water, while the learned reverse process attempts to reconstruct the original structure from noise.

Figure 3.12: The analogy between a diffusion process and the dispersal of color in water.

Figure 3.13 provides a summary of the diffusion processes: the top arrow shows the forward process that adds noise, and the bottom arrow shows the reverse process learned by the model.

Forward diffusion process

Starting from a sample $\mathbf{x}^{(0)}$ drawn from the true data distribution $q\left(\mathbf{x}^{(0)}\right)$, the training process iteratively applies a small Gaussian perturbation at each time step $t = 1, \ldots, T$, forming a Markov chain:

$$q(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}|\mathbf{x}^{(0)}) = \prod_{t=1}^{T} q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}),$$

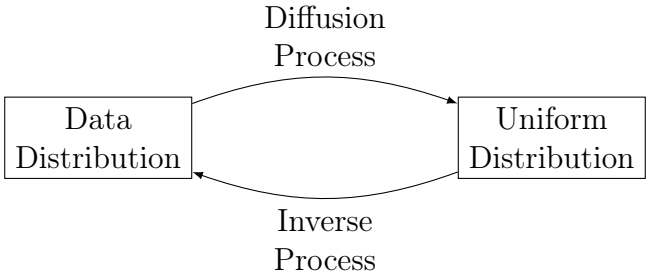


Figure 3.13: Illustration of the diffusion and inverse processes in generative modeling.

where each transition adds Gaussian noise:

$$q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}) = \mathcal{N}(\mathbf{x}^{(t)}; \sqrt{1-\beta_t}\mathbf{x}^{(t-1)}, \beta_t\mathbf{I}).$$

Here, β_t is a variance schedule controlling how much noise is added at each step. As $t \to T$, the samples $\mathbf{x}^{(t)}$ approach an isotropic Gaussian distribution.

Reverse process

The goal is to learn the reversal of this diffusion process, in order to be able to transform noise back into data. This reverse process is also a Markov chain, defined as:

$$p_\theta(\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(T)}) = p(\mathbf{x}^{(T)}) \prod_{t=1}^{T} p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}),$$

where $p(\mathbf{x}^{(T)})$ should be a standard Gaussian and each reverse step is parameterized by a neural network that predicts the mean and variance of the transition:

$$p_\theta(\mathbf{x}^{(t-1)}|\mathbf{x}^{(t)}) = \mathcal{N}(\mathbf{x}^{(t-1)}; \mu_\theta(\mathbf{x}^{(t)}, t), \Sigma_\theta(\mathbf{x}^{(t)}, t)).$$

Training objective

Computing the exact likelihood $p_\theta(\mathbf{x}^{(0)})$ is intractable, since it requires to integrate over all the latent variables $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}$:

$$p_\theta(\mathbf{x}^{(0)}) = \int p_\theta(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}) d\mathbf{x}^{(1:T)}.$$

Therefore, the algorithm optimizes a variational lower bound (the evidence lower bound - ELBO):

$$\begin{aligned}
\log p_\theta(\mathbf{x}^{(0)}) &= \log \int p_\theta(\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)})\, d\mathbf{x}^{(1:T)} \\
&= \log \int \frac{p_\theta(\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(T)})}{q(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}|\mathbf{x}^{(0)})}\, q(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)}|\mathbf{x}^{(0)})\, d\mathbf{x}^{(1:T)} \\
&= \log \mathbb{E}_{q(\mathbf{x}^{(1:T)}|\mathbf{x}^{(0)})} \left[\frac{p_\theta(\mathbf{x}^{(0:T)})}{q(\mathbf{x}^{(1:T)}|\mathbf{x}^{(0)})} \right] \\
&\geq \mathbb{E}_{q(\mathbf{x}^{(1:T)}|\mathbf{x}^{(0)})} \left[\log \frac{p_\theta(\mathbf{x}^{(0:T)})}{q(\mathbf{x}^{(1:T)}|\mathbf{x}^{(0)})} \right],
\end{aligned}$$

where the last step follows from Jensen's inequality. This inequality defines the evidence lower bound (ELBO) on the data log-likelihood:

$$\log p_\theta(\mathbf{x}^{(0)}) \geq \mathbb{E}_q \left[\log \frac{p_\theta(\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(T)})}{q(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(T)} | \mathbf{x}^{(0)})} \right].$$

This corresponds to minimizing the KL divergence between the true forward process and the learned reverse process.

Reparameterization

During training, the reverse transitions are often reparameterized in terms of the noise $\boldsymbol{\epsilon}$ added at each step. The neural network is trained to predict either the clean signal $\mathbf{x}^{(0)}$, the mean μ_θ, or directly the noise $\boldsymbol{\epsilon}$:

$$\mathbf{x}^{(t)} = \sqrt{\alpha_t}\mathbf{x}^{(0)} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}),$$

where $\alpha_t = \prod_{s=1}^{t}(1-\beta_s)$.

Proof. The definition of the forward diffusion process is:

$$\mathbf{x}^{(t)} = \sqrt{1-\beta_t}\mathbf{x}^{(t-1)} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \mathbf{I})$$

We define:

$$\alpha_t := 1 - \beta_t, \quad \bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$$

Then we can write:

$$\begin{aligned}
\mathbf{x}^{(t)} &= \sqrt{1-\beta_t}\mathbf{x}^{(t-1)} + \sqrt{\beta_t}\boldsymbol{\epsilon}_t = \\
&= \sqrt{1-\beta_t}\left(\sqrt{1-\beta_{t-1}}\mathbf{x}^{(t-2)} + \sqrt{\beta_{t-1}}\boldsymbol{\epsilon}_{t-1}\right) + \sqrt{\beta_t}\boldsymbol{\epsilon}_t = \\
&= \sqrt{a_t a_{t-1}}\mathbf{x}^{(t-2)} + \underbrace{\sqrt{a_t(1-a_{t-1})}\boldsymbol{\epsilon}_{t-1}}_{X \sim \mathcal{N}(0, a_t(1-a_{t-1})\mathbf{I})} + \underbrace{\sqrt{1-a_t}\boldsymbol{\epsilon}_t}_{Y \sim \mathcal{N}(0, (1-a_t)\mathbf{I})}
\end{aligned}$$

And since we add two independent random variables that follow Gaussian distributions, the resulting random variable is:

$$Z = X + Y \sim \mathcal{N}(0, \sigma_X^2 + \sigma_Y^2) = \mathcal{N}(0, a_t(1-a_{t-1})\mathbf{I} + (1-a_t)\mathbf{I}) = \mathcal{N}(0, (1-a_t a_{t-1})\mathbf{I})$$

Therefore:

$$\mathbf{x}^{(t)} = \sqrt{a_t a_{t-1}}\mathbf{x}^{(t-2)} + \sqrt{(}1 - a_t a_{t-1})\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I})$$

Continuing this pattern:

$$\mathbf{x}^{(t)} = \sqrt{\bar{\alpha}_t}\mathbf{x}^{(0)} + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$$

This closed-form expression allows us to sample $\mathbf{x}^{(t)}$ directly from $\mathbf{x}^{(0)}$ without simulating all intermediate steps. □

Entropy and information bounds

The work also introduces entropy in the context of diffusion models, to quantify the amount of uncertainty (or disorder) present in the data as the diffusion process evolves. During the forward diffusion process, each step adds Gaussian noise to the data, increasing its entropy.

Mathematically, the total conditional entropy accumulated over the forward process can be expressed as:

$$\mathbb{H}_q(\mathbf{x}^{(T)}|\mathbf{x}^{(0)}) = \sum_{t=1}^{T} \mathbb{H}_q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)}),$$
$$\mathbb{H}_q\big(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}\big) = -\int q\big(\mathbf{x}^{(t)}, \mathbf{x}^{(t-1)}\big) \log q\big(\mathbf{x}^{(t)} \mid \mathbf{x}^{(t-1)}\big)\, d\mathbf{x}^{(t)} d\mathbf{x}^{(t-1)}.$$

where $\mathbb{H}_q(\mathbf{x}^{(t)}|\mathbf{x}^{(t-1)})$ denotes the entropy introduced at each diffusion step due to the added noise. The entropy is related to the following:

- **Quantifying Information Loss:** As entropy increases, the original information is progressively lost. By measuring the total entropy added, we can quantify how much information needs to be recovered in the reverse process.
- **Designing the Reverse Process:** The reverse diffusion process aims to reconstruct the original data from noise. Knowing the entropy added at each step helps in designing neural networks to reverse this process by estimating the necessary information to be retrieved.
- **Thermodynamic Consistency:** Diffusion models draw inspiration from nonequilibrium thermodynamics, and the change of entropy during the forward and reverse processes should align with thermodynamic principles in order to maintain the physical plausibility of the model.

3.4.2 Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models (DDPMs) [29] are latent-variable generative models that learn complex data distributions by reversing a noising process. Figure 3.14 illustrates the forward and reverse diffusion processes. They introduce latent variables $\mathbf{x}_1, \ldots, \mathbf{x}_T$ of the same

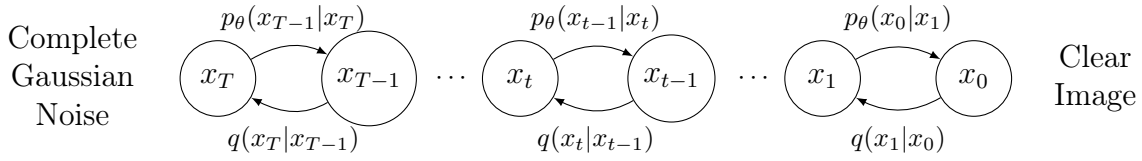


Figure 3.14: Forward and Reverse Diffusion Process Markov Chain.

dimensionality as the data $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, and define the joint probability distribution of all the variables:

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t),$$

where $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and each learned reverse step is

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\big(\mathbf{x}_{t-1};\, \mu_\theta(\mathbf{x}_t, t),\, \Sigma_\theta(\mathbf{x}_t, t)\big).$$

The *forward* diffusion process is a fixed Markov chain that gradually adds Gaussian noise to the data:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}), \quad q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\big(\sqrt{1-\beta_t}\,\mathbf{x}_{t-1},\, \beta_t \mathbf{I}\big).$$

Here $\{\beta_t\}$ is a variance schedule that guides the diffusion process by defining the amount of noise added in in time step. As $t \to T$, $q(\mathbf{x}_t)$ approaches $\mathcal{N}(0, I)$.

Training Objective

Directly maximizing the data likelihood $\log p_\theta(\mathbf{x}_0)$ is intractable, since it requires integrating over all latent variables $\mathbf{x}_{1:T}$:

$$p_\theta(\mathbf{x}_0) = \int p_\theta(\mathbf{x}_{0:T})\, d\mathbf{x}_{1:T}.$$

This integral spans a high-dimensional space, since typically hundreds or thousands of latent variables of the same dimensionality as the data are used, and each reverse transition $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is parameterized by a neural network. As a result, computing this marginal exactly is computationally prohibitive.

To overcome this, DDPMs optimize a variational lower bound (ELBO) on the negative log-likelihood using a fixed forward process $q(\mathbf{x}_{1:T} \mid \mathbf{x}_0)$ and a learned reverse process $p_\theta(\mathbf{x}_{0:T})$. The bound is:

$$\mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_0)\right] \le \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t=1}^{T} \log \frac{p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)}{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}\right] =: L.$$

If we assume each reverse transition has fixed covariance $\Sigma_\theta(\mathbf{x}_t, t) = \sigma_t^2 I$, then the ELBO contribution at step t can be written as

$$\begin{aligned} L_t &= \mathbb{E}_{q(\mathbf{x}_{t-1:t} \mid \mathbf{x}_0)}\Big[\log q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \;-\; \log p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\Big] \\ &= \mathbb{E}_{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}\Big[\underbrace{\mathbb{E}_{q(\mathbf{x}_t \mid \mathbf{x}_{t-1})}\Big[\log q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) - \log p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\Big]}_{\mathrm{KL}\left(q(\mathbf{x}_t \,|\, \mathbf{x}_{t-1}) \,\|\, p_\theta(\mathbf{x}_{t-1} \,|\, \mathbf{x}_t)\right)}\Big] \\ &= \mathbb{E}_{q(\mathbf{x}_{t-1} \mid \mathbf{x}_0)}\Big[\mathrm{KL}\big(q(\mathbf{x}_t \,|\, \mathbf{x}_{t-1}) \,\|\, p_\theta(\mathbf{x}_{t-1} \,|\, \mathbf{x}_t)\big)\Big]. \end{aligned}$$

Since

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}\big(\mu_q,\, \Sigma_q\big), \quad p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\big(\mu_\theta,\, \Sigma_\theta\big),$$

with $\mu_q = \tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0)$, $\Sigma_q = \beta_t I$ and $\Sigma_\theta = \sigma_t^2 I$, the KL divergence has the closed-form

$$\mathrm{KL}\big(\mathcal{N}(\mu_q, \Sigma_q) \,\big\|\, \mathcal{N}(\mu_\theta, \Sigma_\theta)\big) = \frac{1}{2}\Big[\log \frac{\det \Sigma_\theta}{\det \Sigma_q} - d + \mathrm{tr}\big(\Sigma_\theta^{-1}\Sigma_q\big) + (\mu_q - \mu_\theta)^\top \Sigma_\theta^{-1}(\mu_q - \mu_\theta)\Big].$$

Here d is the data dimension. Only the final term depends on θ; the others are constants w.r.t. θ. Moreover, $\Sigma_\theta^{-1} = (1/\sigma_t^2)I$, so

$$(\mu_q - \mu_\theta)^\top \Sigma_\theta^{-1}(\mu_q - \mu_\theta) = \frac{1}{\sigma_t^2}\,\|\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)\|^2.$$

Ignoring θ-independent constants, each L_t reduces to

$$L_t \;\propto\; \frac{1}{2\,\sigma_t^2}\,\mathbb{E}_q\Big[\|\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0)-\mu_\theta(\mathbf{x}_t,t)\|^2\Big].$$

Summing over $t=1,\dots,T$ yields the weighted-squared-error form in the ELBO:

$$L=\sum_{t=1}^{T}L_t \;=\; \sum_{t=1}^{T}\frac{1}{2\,\sigma_t^2}\,\mathbb{E}_q\Big[\|\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0)-\mu_\theta(\mathbf{x}_t,t)\|^2\Big] \;+\; (\text{const.}).$$

Closed-Form Forward Sampling

Because the forward process is constructed from Gaussian transitions, we can express the marginal $q(\mathbf{x}_t \mid \mathbf{x}_0)$ in closed form, in the same way we did in Section 3.4.1:

$$q(\mathbf{x}_t \mid \mathbf{x}_0)=\mathcal{N}\left(\mathbf{x}_t;\sqrt{\bar{\alpha}_t}\mathbf{x}_0,(1-\bar{\alpha}_t)I\right),$$

where $\alpha_t=1-\beta_t$ and $\bar{\alpha}_t=\prod_{s=1}^{t}\alpha_s$. Using this, we can sample $\mathbf{x}_t$ directly from $\mathbf{x}_0$:

$$\mathbf{x}_t=\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},\quad \boldsymbol{\epsilon}\sim\mathcal{N}(0,I).$$

This reparameterization enables efficient training using stochastic gradient descent, since it allows computing expectations over q using samples of $(\mathbf{x}_0,t,\boldsymbol{\epsilon})$ without sequentially calculating the full chain.

Reverse Process Parameterization

In practice, the reverse variances σ_t^2 are fixed (e.g. β_t or $\tilde{\beta}_t$), and the network is tasked with predicting the noise $\boldsymbol{\epsilon}$ at each step. Recall that the ELBO contribution at timestep t reduces (up to constants) to

$$L_{t-1} \;\propto\; \mathbb{E}_q\big[\|\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0)-\mu_\theta(\mathbf{x}_t,t)\|^2\big],$$

where the true posterior mean is

$$\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0)=\frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t-\tfrac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}\Big),\quad \mathbf{x}_t=\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon}.$$

The authors choose the model mean

$$\mu_\theta(\mathbf{x}_t,t):=\frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t-\tfrac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\Big),$$

so that substituting into the squared-error gives

$$\|\tilde{\mu}_t-\mu_\theta\|^2=\frac{\beta_t^2}{\alpha_t(1-\bar{\alpha}_t)}\,\big\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\big\|^2.$$

Absorbing the factor $\beta_t^2/[\alpha_t(1-\bar{\alpha}_t)]$ into the proportionality yields the *noise-prediction loss*:

$$L_{t-1} \;\propto\; \mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\Big[\,\big\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\epsilon},\;t)\big\|^2\Big].$$

Once $\boldsymbol{\epsilon}_\theta$ is trained, sampling $\mathbf{x}_{t-1}$ uses the reparameterization trick:

$$\mathbf{x}_{t-1}=\frac{1}{\sqrt{\alpha_t}}\Big(\mathbf{x}_t-\frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\,\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\Big)+\sigma_t\,\mathbf{z},\quad \mathbf{z}\sim\mathcal{N}(0,I),$$

which remains fully differentiable with respect to θ.

Training Procedure

In each training iteration:

1. Sample a data point $\mathbf{x}_0 \sim q(\mathbf{x}_0)$.
2. Sample a timestep $t \sim \text{Uniform}\{1, \ldots, T\}$.
3. Sample $\boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$ and construct $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}$.
4. Optimize the objective $\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\|^2$ using gradient descent.

This process minimizes the variational bound without needing to compute the intractable marginal likelihood directly.

3.4.3 Score-Based Generative Modeling via Stochastic Differential Equations

Song et al. [6] present a continuous-time framework for generative modeling based on Stochastic Differential Equations (SDEs). Instead of discrete noise steps, they perturb data through a continuous diffusion process and then reconstruct samples by reversing that process with a learned *score function*.

Figure 3.15 illustrates how a clean image is gradually transformed into Gaussian noise via a continuous-time SDE over $t \in [0, T]$.

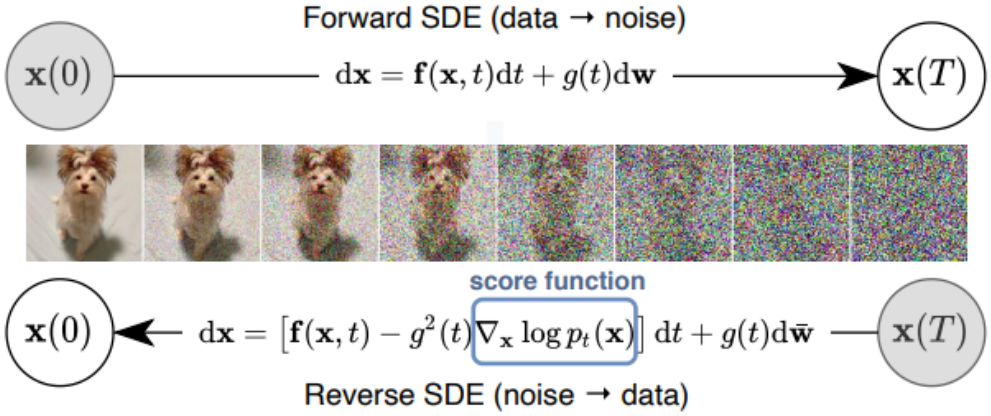


Figure 3.15: Sample trajectory under the SDE (forward and reverse) [6].

Forward and Reverse SDEs

The forward diffusion is governed by:

$$d\mathbf{x} = \mathbf{f}(\mathbf{x},\, t)\, dt + g(t)\, d\mathbf{w},$$

where $\mathbf{w}$ is Brownian motion. This process progressively corrupts data into noise.

The reverse-time SDE, which "undoes" this diffusion, is given by:

$$d\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x})\right] \, dt + g(t) \, d\bar{\mathbf{w}},$$

where $\nabla_{\mathbf{x}} \log p_t(\mathbf{x})$ is the *score function*, which is the gradient of the log-density at time t.

Score Estimation and Sampling

A neural network $s_\theta(\mathbf{x}, t)$ is trained to approximate the score using score matching:

$$\mathbb{E}_{t \sim \text{Uniform}(0,T),\, \mathbf{x} \sim p_t} \left[\lambda(t) \| \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) - s_\theta(\mathbf{x}, t) \|^2 \right],$$

where $\lambda(t) = g(t)^2$ weights the loss appropriately over time. Once s_θ is trained, sampling proceeds by numerically solving the reverse SDE starting from $\mathbf{x}_T \sim \mathcal{N}(0, I)$.

Probability Flow ODE

The *probability flow ODE* is an equivalent deterministic process that matches the marginals of the SDE:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t) - \tfrac{1}{2} g(t)^2 \nabla_{\mathbf{x}} \log p_t(\mathbf{x}).$$

This ODE, shown also in Figure 3.16 allows exact likelihood estimation and can be integrated using ODE solvers, which is more efficient than stochastic SDE sampling.

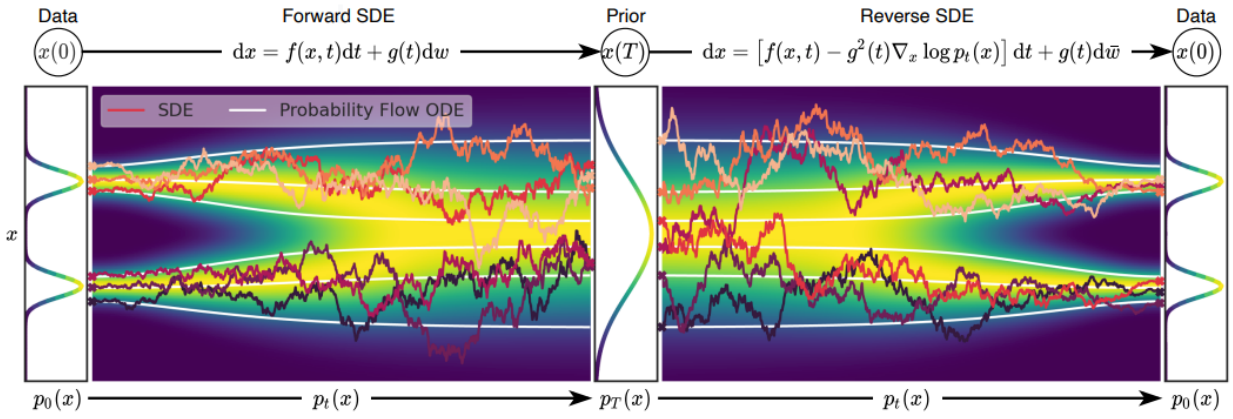


Figure 3.16: Probability flow ODE equivalent to the reverse SDE [6].

Predictor–Corrector Sampling

To improve sample fidelity, the authors propose a Predictor–Corrector (PC) sampler:

- **Predictor**: takes a small step along the reverse SDE.
- **Corrector**: applies several Langevin dynamics steps using the learned score s_θ.

This combination reduces discretization errors and leads to higher-quality samples than using either method alone.

Conditional Sampling and Control

The framework naturally allows conditional generation by augmenting the reverse SDE with an extra term:

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x} \mid \mathbf{y}),$$

which decomposes as $\nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + \nabla_{\mathbf{x}} \log p_t(\mathbf{y} \mid \mathbf{x})$, enabling guided generation.

3.5 Diffusion Models for Time Series Forecasting

While diffusion models are traditionally used for image synthesis tasks, their application has expanded to time series tasks like forecasting, imputation and generation.

We will focus on multivariate time series forecasting which means to predict future values, based on the current and past data. This task is more complex for multiple, correlated variables. Diffusion models, due to their flexibility in capturing complex relations in data with multiple dimensions, outperform the traditional generative methods. In this section, we discuss several diffusion-based models for time series forecasting, focusing on *TimeGrad*.

3.5.1 Problem Formulation

Following the survey paper [30], multivariate time series are denoted as $X^0 = \{x_1^0, x_2^0, \ldots, x_T^0\}$, where $x_i^0 \in \mathbb{R}^D$ are the observations for time point i (initial, unperturbed data). The forecasting goal is to predict future values $X_p^0 = \{x_{t_0}^0, \ldots, x_T^0\}$, based on some context window $X_c^0 = \{x_1^0, x_2^0, \ldots, x_{t_0-1}^0\}$. The joint probability distribution for forecasting is defined as:

$$q(x_{t_0:T}^0 \mid x_{1:t_0-1}^0) = \prod_{t=t_0}^{T} q(x_t^0 \mid x_{1:t_0-1}^0).$$

3.5.2 TimeGrad Model

TimeGrad [7] is an autoregressive model that utilizes Denoising Diffusion Probabilistic Models (DDPMs) [29] to forecast multivariate probabilistic time series. This method combines autoregressive models with the flexibility of diffusion models and achieved state-of-the-art performance in capturing temporal dependencies of high-dimensional data, with complex distributions, in time series forecasting tasks.

Method

The core idea of TimeGrad is to model the conditional distribution of future time steps given past data and the covariates (possible additional information). The model can be expressed as:

$$q_X(\mathbf{x}_{t_0:T} | \mathbf{x}_{1:t_0-1}, c_{1:T}) = \prod_{t=t_0}^{T} q_X(\mathbf{x}_t | \mathbf{x}_{1:t-1}, c_{1:T}),$$

where $\mathbf{x}_t$ is the multivariate vector at time step t and $c_{1:T}$ are some known covariates across time points. TimeGrad models each factor in the product using a conditional denoising diffusion model. The temporal dependencies are encoded using a RNN-based architecture:

$$\mathbf{h}_{t-1} = \text{RNN}_\theta(\text{concat}(\mathbf{x}_{t-1}, \mathbf{c}_t), \mathbf{h}_{t-2}),$$

where $\mathbf{h}_{t-1}$ is the hidden state and $\mathbf{c}_t$ is the covariate at time t. This allows to sample the next step autoregressively:

$$p_\theta(\mathbf{x}_t|\mathbf{h}_{t-1}) \approx q(\mathbf{x}_t \mid \mathbf{x}_{1:t-1}, c_{1:T})$$

With θ representing the weights of both the RNN and the DDPM, which are being optimized simultaneously in the training process.

Training

Training is performed by sampling random windows of time series data and splitting them into context and prediction windows. The training algorithm is:

1. Sample timestep $n \sim 1, \ldots, N$
2. Sample $\epsilon \sim \mathcal{N}(0, \mathbf{I})$
3. Form $\tilde{x}_t = \sqrt{\bar{a}_n} x_t + \sqrt{1 - \bar{a}_n}\epsilon$
4. Train to reduce the loss function.

The objective is the minimization of the negative log-likelihood over the prediction window, given the context window:

$$\sum_{t=t_0}^{T} - \log p_\theta(\mathbf{x}_t|\mathbf{h}_{t-1}).$$

The loss function, follows the DDPM's logic:

$$\mathbb{E}_{\mathbf{x}_t,\boldsymbol{\epsilon},n}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_n}\mathbf{x}_t + \sqrt{1-\bar{\alpha}_n}\boldsymbol{\epsilon}, \mathbf{h}_{t-1}, n)\|^2\right] = \mathbb{E}_{\mathbf{x}_t,\boldsymbol{\epsilon},n}\left[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\tilde{x}_t, \mathbf{h}_{t-1}, n)\|^2\right],$$

where $\boldsymbol{\epsilon}$ is the noise sampled from a standard Gaussian distribution, and n is the noise level. The noise index is encoded with Fourier positional embeddings and $\boldsymbol{\epsilon}_\theta$ is a neural network's prediction of the noise for each time step.

Inference

Once ϵ_θ and the RNN have been trained, TimeGrad generates future forecasts autoregressively as follows.

1. **Initialize:** At the last observed time $t_0 - 1$, compute the hidden state

$$\mathbf{h}_{t_0-1} = \mathrm{RNN}_\theta\big(\mathbf{x}_{1:t_0-1}, c_{1:t_0-1}\big).$$

2. **For each forecast step $t = t_0, \ldots, T$:**

   (a) *Sample initial noise*:

$$\mathbf{x}_t^{(N)} \sim \mathcal{N}\big(\mathbf{0}, I\big),$$

   where N is the total number of noise-levels used in training.

   (b) *Annealed Langevin Dynamics (Predictor–Corrector).*
   For $n = N, N-1, \ldots, 1$:

i. **Predictor step:** Perform one reverse-diffusion update

$$\mathbf{x}_t^{(n*)} = \mathbf{x}_t^{(n)} - \frac{\beta_n}{\sqrt{1-\bar{\alpha}_n}}\,\epsilon_\theta\big(\mathbf{x}_t^{(n)},\, \mathbf{h}_{t-1},\, n\big) \;+\; \sqrt{\sigma_n^2}\,\mathbf{z}, \quad \mathbf{z}\sim\mathcal{N}(0,I),$$

where $\bar{\alpha}_n = \prod_{i=1}^{n}(1-\beta_i)$ and σ_n is the fixed reverse variance.

ii. **Corrector step:** Apply K steps of Langevin dynamics to refine the sample:

$$\mathbf{x}_t^{(n,k+1)} = \mathbf{x}_t^{(n,k)} + \eta\, s_\theta\big(\mathbf{x}_t^{(n,k)},\, \mathbf{h}_{t-1},\, n\big) + \sqrt{2\eta}\,\mathbf{u}, \quad \mathbf{u}\sim\mathcal{N}(0,I),$$

for $k = 0,\ldots,K-1$, with step-size η, and initialize $\mathbf{x}_t^{(n,0)} = \mathbf{x}_t^{(n*)}$. Set $\mathbf{x}_t^{(n-1)} = \mathbf{x}_t^{(n,K)}$.

(c) *Finalize sample.* After completing $n = 1$, we obtain $\widehat{\mathbf{x}}_t = \mathbf{x}_t^{(0)}$, which is the forecast for time t.

(d) *Update hidden state.* Incorporate the new forecast into the RNN:

$$\mathbf{h}_t = \mathrm{RNN}_\theta\big(\widehat{\mathbf{x}}_t,\, c_t,\, \mathbf{h}_{t-1}\big).$$

3. **Repeat** for all $t_0 \le t \le T$. The result is a set of trajectories $\{\widehat{\mathbf{x}}_{t_0:T}\}$. By running this procedure multiple times (with different random seeds), we obtain an ensemble of sample paths that approximate the joint predictive distribution.

This *Predictor–Corrector* algorithm, which alternates between a single reverse-diffusion "predictor" step and multiple Langevin "corrector" steps, reduces discretization errors and yields high-quality probabilistic forecasts.

Results

TimeGrad outperforms various state-of-the-art models on almost all real-world datasets it was tested on: Exchange, Solar, Electricity, Traffic, Taxi, and Wikipedia. The evaluation metric is the Continuous Ranked Probability Score (CRPS), which measures the quality of probabilistic forecasts (see Section 3.2.4 for more information and intuitive explanation).

Table 3.1 shows a comparison of CRPS scores between TimeGrad and several classical models (VAR, VAR-Lasso, GARCH) and modern deep learning models (Vec-LSTM, GP-Copula, Transformer-MAF). TimeGrad outperforms the other methods on most datasets including the ones that are high-dimensional (Electricity and Traffic), showcasing its flexibility, scalability and ability to model complex temporal dependencies.

3.5.3 ScoreGrad

ScoreGrad [8] is another diffusion-based model for time series forecasting, which extends the diffusion process to continuous time using Stochastic Differential Equations (SDEs). The reverse process for ScoreGrad is modeled by a time-reversed SDE:

$$dx_t = \big[f(x_t,k) - g(k)^2\nabla_{x_t}\log q_k(x_t \mid h_t)\big]\,dk + g(k)dw,$$

where the conditional score $\nabla_{x_t}\log q_k(x_t \mid h_t)$ is approximated by a neural network $s_\theta(x_t,h_t,k)$:

$$L_t(\theta) = E_{k,x_0^t,x_k^t}\left[\delta(k)\left\|s_\theta(x_k^t,h_t,k) - \nabla_{x_t}\log q(x_t^0 \mid x_0^t)\right\|^2\right].$$

ScoreGrad applies a predictor-corrector sampler [6] for forecasting, which refines the time-reversed SDE sampling process.

Table 3.1: Test set CRPSsum comparison (lower is better) of models on six real-world datasets. TimeGrad [7] establishes state-of-the-art performance across most datasets.

Method	Exchange	Solar	Electricity	Traffic	Taxi	Wikipedia
VAR	0.005	0.83	0.039	0.29	0.292	3.4
VAR-Lasso	0.012	0.51	0.025	0.15	-	3.1
Vec-LSTM	0.008	0.391	0.025	0.087	0.506	0.133
Transformer-MAF	0.005	0.301	0.021	0.056	0.179	0.063
TimeGrad	0.006	0.287	0.021	0.044	0.114	0.048

Model Architecture

ScoreGrad has two main modules. The *Time Series Feature Extraction* module extracts features from historical time series data up until time step $t-1$ and at each time step, a feature vector $\mathbf{F}_t$ is updated:

$$\mathbf{F}_t = R(\mathbf{F}_{t-1}, \mathbf{x}_{t-1}, \mathbf{c}_{t-1}),$$

where R can be any sequential model such as RNNs and GRUs. The *Conditional SDE-Based Score Matching* module, at each time step t, utilizes the conditional score function $\nabla_{\mathbf{x}} \log p(\mathbf{x}_t|\mathbf{F}_t)$ to model the reverse-time SDE:

$$d\mathbf{x}_t = [f(\mathbf{x}_t, t) - g(t)^2 \nabla_{\mathbf{x}} \log p(\mathbf{x}_t|\mathbf{F}_t)]dt + g(t)dw,$$

where $f(\mathbf{x}_t, t)$ and $g(t)$ are the drift and diffusion coefficients, respectively, and w is a Wiener process. The conditional score function is implemented using a neural network inspired by WaveNet and DiffWave [49, 50].

Training

The training process involves optimizing a continuous form of the score matching loss. For each time step t, the loss is defined as:

$$\mathcal{L}_t(\theta) = \mathbb{E}_{t_s}\left[\lambda(t_s)\mathbb{E}_{\mathbf{x}_{0t},\mathbf{x}_{t_s}|\mathbf{x}_{0t}}\left[\|s_\theta(\mathbf{x}_{t_s}, \mathbf{F}_t, t_s) - \nabla_{\mathbf{x}_{t_s}} \log p(\mathbf{x}_{t_s}|\mathbf{x}_{0t})\|_2^2\right]\right],$$

where $\lambda(t_s)$ is a weighting function and t_s is randomly sampled from the interval $[0, T_s]$ and the total loss is the average of the loss values across all time steps:

$$\mathcal{L}(\theta) = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}_t(\theta).$$

Sampling and Prediction

The prediction is derived by iteratively sampling from the reverse-time SDE. Beginning with an initial sample $\mathbf{x}_{T_s}$ from the target distribution, ScoreGrad solves the reverse SDE (with numerical solvers), applying a Predictor-Corrector method to iteratively refine the sample and yield the predicted value x_t.

Results

ScoreGrad was evaluated on the same six real-world datasets as TimeGrad using the CRPS evaluation metric. The authors examined three types of SDEs: VP SDE (Variance Preserving SDE – designed so that the variance of the data is preserved through the forward diffusion process), VE SDE (Variance Exploding SDE – the variance of the data increases significantly (explodes) over time as noise is added in the forward diffusion process), and sub-VP SDE (a hybrid approach that aims to preserve variance, similar to VP SDEs, but with a noise schedule limited by the VP process). ScoreGrad achieved state-of-the-art results on these datasets, slightly better than TimeGrad, shown in Table 3.2.

Table 3.2: Comparison of CRPS scores for ScoreGrad [8] and other methods on real-world datasets. Lower is better.

Method	**Exchange**	**Solar**	**Electricity**	**Traffic**	**Taxi**	**Wikipedia**
VAR	0.005	0.840	0.038	0.291	-	-
Lasso-VAR	0.011	0.521	0.026	0.153	-	3.142
VES	0.006	0.896	0.883	0.362	-	-
GARCH	0.023	0.884	0.193	0.376	-	-
KVAE	0.014	0.339	0.050	0.108	-	0.095
Vec-LSTM (lowrank-Copula)	0.007	0.319	0.064	0.103	0.326	0.241
Vec-LSTM (ind-Scaling)	0.008	0.391	0.025	0.087	0.506	0.133
GP-Scaling	0.009	0.368	0.022	0.079	0.183	1.483
GP-Copula	0.007	0.337	0.025	0.078	0.208	0.086
Transformer-MAF	0.005	0.301	0.020	0.056	0.179	0.063
TimeGrad	0.006	0.287	0.020	0.049	0.114	0.050
ScoreGrad (VP SDE)	0.006	0.268	0.019	0.043	0.102	0.041
ScoreGrad (VE SDE)	0.007	0.277	0.019	0.037	0.104	0.046
ScoreGrad (sub-VP SDE)	0.006	0.256	0.019	0.041	0.101	0.043

3.5.4 Diffusion, Denoise and Disentanglement BVAE (D3VAE)

Although the ideas and methods of D3VAE [31] will not be examined in this thesis, it is worth mentioning. D3VAE addresses the challenges of predicting noisy and short time series by using a coupled diffusion process for both the context and prediction windows. It utilizes a bidirectional variational autoencoder (BVAE) and a denoising score matching module to improve predictions. The model's training goal is to minimize a combination of four losses: KL divergence, denoising score matching, latent disentanglement and the mean squared error between the prediction and actual values:

$$\mathcal{L} = \psi D_{\mathrm{KL}}(q(x_0^{t_0:T}) \mid p_\theta(x_0^{t_0:T})) + \lambda \mathcal{L}_{\mathrm{DSM}} + \gamma \mathcal{L}_{\mathrm{TC}} + \mathcal{L}_{\mathrm{MSE}}.$$

Model Architecture The structure of D3VAE consists of four main components.

Coupled Diffusion Process D3VAE uses a forward diffusion process that adds Gaussian noise to both the input and output time series simultaneously:

$$q(\mathbf{X}_{1:T}|\mathbf{X}_0) = \prod_{t=1}^{T} q(\mathbf{X}_t|\mathbf{X}_{t-1}), \quad q(\mathbf{X}_t|\mathbf{X}_{t-1}) = \mathcal{N}(\mathbf{X}_t; \sqrt{1-\beta_t}\mathbf{X}_{t-1}, \beta_t\mathbf{I})$$

where $\mathbf{X}_0$ is the original input time series, and β_t is a variance schedule controlling the level of noise. The same process is applied to the target series $\mathbf{Y}_0$, using a variance schedule β'_t.

This process augments with noise both imput and target series, thus allowing the model to learn robust representations in the presence of uncertainty.

Bidirectional Variational Autoencoder (BVAE) D3VAE employs BVAE as the reverse process to reconstruct the clean target series from the noisy diffused one. The latent variable $\mathbf{Z}$ is modeled in a multivariate way and follows the generative process:

$$p_\theta(\mathbf{Y}|\mathbf{Z}) = p_\theta(\mathbf{Y}_r|\mathbf{Z})p_\theta(\epsilon_Y|\mathbf{Z})$$

where $\mathbf{Y}_r$ is the ideal clean part of the generated target series, and ϵ_Y represents the noise. The VAE can capture complex temporal dynamics and maintain tractability.

Denoising Score Matching The diffusion process introduces noise that degrades the quality of the generated target series. To resolve this, D3VAE uses denoising score matching (DSM), which cleans the generated sequences by minimizing the difference between the noisy generated data and the clean data:

$$L_{\text{DSM}}(\zeta, t) = \mathbb{E}_{p_{\sigma_0}(\mathbf{Y}_t, \mathbf{Y})}\left[\|\mathbf{Y} - \mathbf{Y}_t + \sigma_0^2 \nabla_{\mathbf{Y}_t} E(\mathbf{Y}_t; \zeta)\|^2\right]$$

where $\mathbf{Y}_t$ is the noisy generated series at time t, and $E(\mathbf{Y}_t; \zeta)$ is an energy function. This way, the model moves the generated data closer to the clean target.

Latent Variable Disentanglement An extra step to enhance the interpretability and stability of the forecasts, D3VAE disentangles the latent multivariate variables $\mathbf{Z}$. This is done by minimizing the total correlation (TC), which is a measure of the dependency between different dimensions of the latent space:

$$TC(\mathbf{z}_i) = D_{\text{KL}}(p_\phi(\mathbf{z}_i)\|\bar{p}_\phi(\mathbf{z}_i)), \quad \bar{p}_\phi(\mathbf{z}_i) = \prod_j p_\phi(\mathbf{z}_{i,j})$$

By doing so, D3VAE makes different dimensions of the latent space correspond to different temporal patterns, like trend or seasonality.

Training Objective

As mentioned, the training loss consists of four components:

$$L = \psi \cdot D_{\text{KL}}(q(\mathbf{Y}_t)\|p_\theta(\mathbf{Y}_t)) + \lambda \cdot L_{\text{DSM}}(\zeta, t) + \gamma \cdot L_{\text{TC}} + L_{\text{MSE}}(\mathbf{Y}_t, \hat{\mathbf{Y}}_t)$$

where L_{MSE} represents the mean squared error between the generated and true target series, and ψ, λ, γ are trade-off parameters balancing the different loss components.

3.5.5 Other Diffusion-Based Time Series Models

The work [30] discusses three additional models for time series forecasting. Although their ideas will not be applied to this work, we mention them briefly.

Diffusion Schrödinger Bridge (DSB)

DSB [32] applies a Schrödinger bridge framework to diffusion processes for multivariate time series forecasting. This way it aims to learn a probabilistic time series model by constructing a bridge between two distributions over time, similar to the diffusion process. The Schrödinger bridge formulation can help learn complex temporal dependencies, due to its flexibility. DSB showcased state-of-the-art performance in tasks like long-term traffic and energy demand forecasting.

Diffusion Spatio-Temporal Graph (DiffSTG)

DiffSTG [33] extends diffusion models for spatio-temporal time series, in order to handle time series data with spatial dependencies. Using a spatio-temporal graph (STG) to represent the data, DiffSTG can model spatial and temporal dependencies via graph diffusion. This model is particularly useful for applications where spatial relationships between time series variables are important, and the results show that DiffSTG has state-of-the-art results in traffic and air quality datasets.

Graph Convolutional Recurrent Denoising Diffusion (GCRDD)

GCRDD [34] combines graph convolutional networks (GCNs) and recurrent neural networks (RNNs) within the diffusion process to model multivariate time series with temporal and spatial dependencies. The model applies denoising diffusion in a graph-structured latent space. The GCN captures spatial relations and the RNN captures temporal dependencies. GCRDD also achieves state-of-the-art forecasting accuracy.

3.6 Reinforcement Learning

3.6.1 Introduction to Reinforcement Learning

Definitions and Overview

Reinforcement Learning (RL) is an approach that enables an *agent* to learn by interacting with its *environment*. An agent is the decision-making entity that interacts with the environment acting on it, based on the environment's current state. The environment is everything that the agent interacts with. It is the external system that provides the system's state for the agent, gets acted upon by the agent and responds by providing a new state and information about the agent's actions (a reward signal that will be discussed thoroughly later). The environment's response can be either deterministic; the same action on the same state always returns the same outcome, or stochastic; the same action in the same state could result in different outcomes if some context is different [35]. An example of an entirely determinictic environment is a game of chess; the rules are fixed and there is no randomness involved, so a legal move by a player (agent) given a piece configuration (state) will result in one and only new piece configuration (new state), whereas an example of a stochastic environment would be the navigation of a robot in a room with slippery floor or moving obstacles; even if the robot makes the same series of actions in two different times, the new states it gets in return may vary. In general, the goal of RL is to make optimal decisions through trial and error or using a model of the system's (environment's) dynamics, by maximizing the quality of its actions, which is encoded as a numerical reward signal. RL agents learn from the consequences of their actions rather than from a set of labeled data.

Reinforcement learning is distinct from supervised and unsupervised learning. In supervised learning, a model learns from a set of labeled examples, each one of them consisting of a situation and the corresponding correct action, which is then used by the model to learn to generalize to new situations. RL differs in that it does not rely on labeled examples; the agent must learn from the feedback it receives by acting on the environment. Unsupervised learning, mainly aims to find patterns or structures in unlabeled data. While RL also does not use explicit labels, its objective is not to learn hidden structures in data but to maximize a reward signal based on the agent's actions. Thus, reinforcement learning is often regarded as a third machine learning paradigm.

History of Reinforcement Learning

The field of Reinforcement Learning has emerged from the combination of two distinct research areas. The one significant area is *behavioral psychology*, where the mechanisms of learning through interaction with the environment were first explored. The "Law of Effect," proposed by Thorndike [51], provided a foundation on how consequences shape behavior, suggesting that satisfying outcomes associate actions and situations, through feedback.

Independently, the *optimal control* area was developing mathematical techniques for sequential decision-making. The concept of *dynamic programming*, introduced by Bellman [38] in the mid-20th century, was a powerful approach to solving problems where the goal is to find an optimal sequence of actions over time, particularly when a model of the system's behavior is available. Initially it focused on control, however the iterative nature of dynamic programming later led to learning-based approaches..

A crucial step in the evolution of RL was the formalization of the learning problem within

the framework of *Markov Decision Processes* (MDPs). This served as a common language and mathematical structure for understanding agent-environment interaction. Building upon this foundation, the first RL algorithms were developed towards the end of the 20th century. *Temporal Difference learning* (TD learning), introduced by Sutton [36], allowed the agents to learn predictions from experience by examining the difference between successive estimates. *Q-learning*, developed by Watkins and Dayan [37], provided a method for learning optimal action-value functions solely from interaction, without requiring a model of the environment.

Field progress led to the need of handling increasingly complex problems with large state and action spaces. *Function approximation* techniques, mostly developed during the 1980s and 1990s, played a critical role in addressing this. However, the breakthrough with the largest impact in tackling high-dimensional spaces, was *deep learning* in the 21st century. The integration of deep neural networks with RL techniques led to *Deep Reinforcement Learning*, with remarkable capabilities in domains which were previously considered intractable, such as mastering video games and controlling sophisticated robots [52].

Learning from Interaction

As introduced in the definitions, the fundamental idea in reinforcement learning is learning from interaction. RL models try to mimic the way humans and animals learn from their environments. The RL agent interacts with an environment in a loop: the agent analyzes the state of the environment, takes an action and then receives feedback in the form of a reward signal and the new state it transitioned to. The agent's goal is to learn a policy, a mapping from states to actions, that maximizes the cumulative reward over time, which translates to the agent approximating its goal.

A nice way to formalize a RL problem, as discussed briefly previously, is using Markov Decision Processes (MDPs). The MDPs are mathematical models that describe the dynamics of the environment with respect to states, actions and rewards. This formalization enables the agent to consider the long-term reward of its actions, which is necessary to act optimally. Through trial and error, the agent explores different actions by traversing the MDP, in order to discover the ones that yield the highest rewards. Delayed reward is the idea that an action may not only affect the immediate reward but also the future states and rewards.

In many real-world scenarios, the agent does not have access to the complete state of the environment; rather, it receives partial, noisy observations that capture part of the system dynamics. Partially Observable Markov Decision Processes (POMDPs) provide a framework for formulating such problems, extending classical MDPs. In a POMDP, the agent receives observations that are probabilistically related to the true, hidden state, and maintains a belief state, which is a probability distribution over the possible states. This belief is updated with new observations, allowing the agent to handle both immediate uncertainties and the evolution of the system. The goal of the agent becomes optimizing its policy over the belief states.

3.6.2 Core Concepts of Reinforcement Learning

Exploration vs. Exploitation

A challenge in reinforcement learning is balancing exploration and exploitation. Exploration means to try new actions to discover their potential rewards, while exploitation refers to selecting the actions that the agent has already found to be most rewarding. For maximizing the total reward, the two strategies must be balanced. Too much exploration leads to suboptimal behavior,

as the agent continually tries unexplored actions instead of exploiting known good actions. Conversely, too much exploitation prevents the agent from discovering better actions that it has not tried yet.

Agents, Environments, and Interactions

In reinforcement learning, the interaction between an agent and its environment is formalized within the framework of a Markov Decision Process, as discussed before. The environment is characterized by a continuous or discrete set of possible states, denoted as $\mathcal{S}$, and a corresponding set of available actions, $\mathcal{A}$. The *state space* $\mathcal{S}$ encompasses all possible configurations that the environment can be in, and each state should contain complete information about the current situation, for an agent to make a decision (this is an idealized assumption of the MDP formulation. The Partial Observable MDP framework gives a solution for real world environments). The *action space* $\mathcal{A}$ comprises all actions the agent can make at any given state. In practice, the available actions may depend on the particular state, so we often denote the set of admissible actions in state s by $\mathcal{A}(s)$.

At each discrete time step t, the agent receives an observation of the environment, which, under full observability, is the state $s_t \in \mathcal{S}$. Based on this state, the agent selects an action $a_t \in \mathcal{A}(s_t)$ according to its *policy* π. This policy, which may be either deterministic (where each state maps to a specific action, $\pi(s) = a$) or stochastic (where $\pi(a|s)$ represents the probability of choosing action a given state s), is the guiding principle for decision-making.

After executing the chosen action a_t, the environment transitions to a new state s_{t+1} following the *transition probability function* $P(s_{t+1} \mid s_t, a_t)$. In deterministic environments, this transition is a fixed mapping, whereas in stochastic environments it is characterized by uncertainty. Simultaneously, the agent is provided with a *scalar reward* $r_t = R(s_t, a_t)$, which reflects the immediate benefit (or cost) of taking action a_t in state s_t.

The ultimate objective of the agent is to maximize its cumulative reward over time, called *return*. Mathematically, the return G_t from time t onward is defined as

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k},$$

where $\gamma \in [0, 1]$ is the discount factor, determining the relative importance of immediate versus future rewards. This way, the agent learns to balance its actions over time, accounting for both short-term and long-term gains.

Policies

A *policy* $\pi(a|s)$ defines the agent's strategy, mapping states to actions. A policy can be either *deterministic*, where each state is mapped to a specific action, $\pi(s) = a$, or *stochastic*, which provides a probability distribution over actions for each state, $\pi(a|s) = P(a|s)$. The goal is to find an optimal policy π^* that maximizes the expected return.

Value Functions

Value functions estimate how good a state (or state-action pair) is, regarding the expected cumulative reward. Value functions can be perceived from two –ultimately equivalent– angles.

The first way to view them is the *State-Value Function* $V^\pi(s)$, which gives the expected return starting from state s, following policy π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s].$$

The second way is the *Action-Value Function* $Q^\pi(s, a)$, which gives the expected return starting from state s, taking action a and then following policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a].$$

The Bellman equation is fundamental in RL as it provides a recursive relationship for these value functions. For the state-value function V^π, the Bellman equation is:

$$V^\pi(s) = \sum_{a\in\mathcal{A}(s)} \pi(a|s)\left(R(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s, a)V^\pi(s')\right),$$

where s' is the next state after taking action a from state s. This can equivalently be written in expectation form as:

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

Similarly, for the action-value function $Q^\pi(s, a)$, the Bellman equation is given by:

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'\in\mathcal{S}} P(s'|s, a) \sum_{a'\in\mathcal{A}(s')} \pi(a'|s')\, Q^\pi(s', a').$$

And, in expectation form, it is:

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

Furthermore, the state-value and action-value functions are related:

$$V^\pi(s) = \sum_{a\in\mathcal{A}(s)} \pi(a|s)\, Q^\pi(s, a).$$

Challenges of Continuous Spaces in Reinforcement Learning

In many real-world problems the state and action spaces are continuous, thus have infinitely many possible states and actions. There are various related challenges. First of all, there are *Infinite State-Action Combinations*; the agent must approximate the value functions and policies since storing and computing the values of all state-action pairs is impossible. Furthermore, there are *Optimization Challenges* in choosing optimal actions and calculating value functions for continuous spaces. Another challenge is *Exploration*, where efficiently exploring continuous spaces is by far more challenging than exploring discrete spaces due to the infinite number of possible states and actions. Moreover, achieving *Sample Efficiency* is also difficult, as the agent has to learn efficiently from limited experience, since it is infeasible to explore all possible states and actions. In order to handle these challenges, RL algorithms use *function approximation*, *policy gradients*, *actor-critic* architectures, etc. Below, we re-define the key concepts of RL, for continuous spaces this time.

Agents, Environments, and Interactions for Continuous Spaces

In continuous domains the *state space* $S \subset \mathbb{R}^{D_s}$ and the *action space* $A \subset \mathbb{R}^{D_a}$ are defined as subsets of Euclidean spaces, where D_s and D_a denote the dimensionalities of these spaces, respectively. A *state* $s_t \in S$ is represented by a continuous vector that encapsulates the configuration of the environment at time t, while an *action* $a_t \in A$ is a continuous vector that the agent selects in order to affect the environment. The *reward function* $r_t = R(s_t, a_t, s_{t+1})$ delivers a scalar feedback based on both the current state s_t and the action a_t taken, along with the subsequent state s_{t+1}. The return, defined as

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

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

Analogous to the discrete case, an agent interacts with its environment in discrete time steps even when the state and action spaces are continuous. At each time step t, the agent observes the current state $s_t \in \mathcal{S}$ and selects an action a_t from the set of admissible actions $\mathcal{A}(s_t)$. After the agent executes the action a_t, the environment evolves to a new state according to a stochastic process that reflects both the predictable dynamics and inherent uncertainties of the system:

$$s_{t+1} = P(s_t, a_t).$$

Similarly, the reward obtained from a transition is given by a function $R(s_t, a_t, s_{t+1})$:

$$r_{t+1} = R(s_t, a_t, s_{t+1}).$$

In addition, one may express the probability of transitioning to a particular subset $S' \subseteq \mathcal{S}$ as:

$$P(s_{t+1} \in S' \mid s_t = s, a_t = a) = \int_{S'} P(s, a, s')\, ds',$$

thus quantifying the uncertainty in the system dynamics over continuous spaces.

Policies in Continuous Spaces

Like in discrete spaces, a policy π, is a mapping from states to actions. This mapping may be deterministic, with $\pi(s) = a$ where each state s is assigned a specific action a, or stochastic, where the policy is characterized by a probability density function $\pi(a|s) = p(a|s)$ over the action space given a state. In continuous spaces, *policy gradient* methods are used to directly optimize the parameters of the policy by following the gradient of the expected return.

Value Functions in Continuous Spaces

Value functions are an estimation of the expected return from a state (or a state-action pair). For continuous spaces these functions are typically approximated using parameterized function approximators, such as neural networks. The state-value function $V^\pi(s)$ is defined as the expected return starting from state s under policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s \right],$$

while the action-value function $Q^\pi(s,a)$ estimates the expected return from taking action a in state s followed by policy π:

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s, a_t = a \right].$$

The Bellman Equation in Continuous Spaces

In continuous spaces the Bellman equation is expressed using integrals, instead of summations. For the state-value function, the Bellman equation can be written as:

$$V^\pi(s) = \int_{\mathcal{A}} \pi(a|s) \int_{\mathcal{S}} P(s,a,s') \left[R(s,a,s') + \gamma V^\pi(s') \right] ds'\, da,$$

representing an expectation over the actions selected by the policy and the subsequent states produced by the transition function. Similarly, the action-value function satisfies the relation:

$$Q^\pi(s,a) = \int_{\mathcal{S}} P(s,a,s') \left[R(s,a,s') + \gamma \int_{\mathcal{A}} \pi(a'|s') Q^\pi(s',a')\, da' \right] ds'.$$

For an optimal policy π^*, the Bellman optimality equations become:

$$V^*(s) = \max_a \int_{\mathcal{S}} P(s,a,s') \left[R(s,a,s') + \gamma V^*(s') \right] ds'$$

and:

$$Q^*(s,a) = \int_{\mathcal{S}} P(s,a,s') \left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a') \right] ds'.$$

3.6.3 Solving Markov Decision Processes

Approaches to Solve Discrete MDPs

In many reinforcement learning problems the environment is modeled as a Markov Decision Process (MDP) with discrete state and action spaces. At each discrete time step t, the agent observes the current state $s_t \in \mathcal{S}$ and selects an action a_t from the set $\mathcal{A}(s_t)$, usually depending on the state. When an action is executed, the environment transitions to a new state s_{t+1} according to the transition probability function $P(s_{t+1} \mid s_t, a_t)$, and the agent receives a scalar reward $r_t = R(s_t, a_t)$. The ultimate goal is to find a policy π (a mapping from states to actions) that maximizes the expected cumulative reward (or return) defined as

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

where $\gamma \in [0,1]$ is the discount factor that governs the trade-off between immediate and future rewards.

Because the state and action spaces in discrete MDPs are finite, it is possible –theoretically– to represent the value functions (which estimate the expected return) in tabular form. However, as the size of the state space grows, even discrete problems can become demanding, thus require approximate representations and efficient algorithms to search for the optimal policy.

Key Algorithms for Discrete MDPs

Policy Iteration The classical approach for solving discrete MDPs is dynamic programming (DP), which makes use of the Bellman equations to recursively compute optimal value functions and policies. One common method is *policy iteration*, where the agent iteratively alternates between evaluating the current policy and then improving it. In the policy evaluation phase, the value function $V^\pi(s)$ is computed by solving the Bellman expectation equation:

$$V^\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(a|s) \sum_{s' \in \mathcal{S}} P(s'|s,a) \Big[R(s,a) + \gamma\, V^\pi(s') \Big],$$

which provides the expected cumulative reward following policy π. Once the value function is determined, the policy is updated by choosing for each state the action that maximizes the expected return:

$$\pi'(s) = \arg\max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s,a) \Big[R(s,a) + \gamma\, V^\pi(s') \Big].$$

This two-step process is repeated until the policy converges to an optimal policy π^*.

Value Iteration An alternative DP method is *value iteration*, which iteratively updates the value function using the Bellman optimality equation:

$$V(s) = \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}} P(s'|s,a) \Big[R(s,a) + \gamma\, V(s') \Big].$$

Once the value function V^* converges, the optimal policy can be extracted by choosing the action that achieves the maximum value for each state.

Monte Carlo Methods If a model of the environment is not available, *Monte Carlo* methods can be used. They estimate the value function by averaging the observed returns over sample episodes. For instance, in the first-visit Monte Carlo approach the value of state s is estimated as

$$V(s) = \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_{t_i},$$

where $N(s)$ is the number of episodes in which state s is encountered for the first time, and G_{t_i} is the return following the first occurrence of s in each episode. An alternative is the every-visit Monte Carlo method which averages the returns over all visits to state s. A drawback of Monte Carlo methods is that a full episode is required to update the estimation of the value function. Thus, in long episodes a huge amount of steps is needed to approach the value function, while in environments where the whole training lasts one single episode (of infinite period), Monte Carlo methods are of no use. In the last cases, the following algorithm can be useful.

Monte Carlo simulation has two forms. The *first-visit* variant records, for each state s, the return following its first occurrence in an episode. The average of these first-visit returns over many episodes provides an estimate of the state's value. Formally, if $N(s)$ denotes the number of episodes in which s appears for the first time at time steps $t_1, t_2, \ldots, t_{N(s)}$, and G_{t_i} is the corresponding return from t_i onward, then the first-visit estimate is

$$V(s) \;=\; \frac{1}{N(s)} \sum_{i=1}^{N(s)} G_{t_i}.$$

On the other hand, the *every-visit* method does not restrict attention to only the first occurrence; instead it averages the returns following each time the state s is visited. If $M(s)$ is the total count of visits to s across all episodes, at times $t_1, \ldots, t_{M(s)}$, then

$$V(s) \;=\; \frac{1}{M(s)} \sum_{i=1}^{M(s)} G_{t_i}.$$

Both variants converge to the true value function given enough iterations, but each has trade-offs in bias and variance. The *first-visit* method tends to be unbiased because each return following the initial visit to a state within an episode is an independent sample of the expected future reward from that point, because it is not influenced by any subsequent events in the same episode that might be correlated with revisiting the state. The drawback of this method is that it suffers from higher variance as it uses fewer data points for the estimate – only one return per state per episode. Conversely, the *every-visit* method, averages the returns following every visit to a state, typically yields estimates with lower variance, since the more samples used result in the randomness of individual episodes affecting the estimate less. A potential downside of this approach is the slight bias it adds. This bias occurs if revisiting a state within the same episode is not independent of the returns experienced. For example, this occurs when the agent's policy changes upon revisiting a state, or if the environment's dynamics change. Then, averaging all returns might skew the estimate of the value of initially entering that state. However, in many practical scenarios, especially where the policy is stationary, this bias is often minimal.

A noteworthy drawback of Monte Carlo methods is their requirement to see the complete return of an episode before making any update. In tasks with very long—or nonterminating—episodes, the learning is delayed substantially. When episodes are infinite, Monte Carlo updates are infeasible, since the return G_t can never be observed. In such occasions, one may use temporal-difference methods (described below), which update value estimates online without waiting for an episode to end.

Temporal Difference Learning Temporal difference (TD) learning methods combine the advantages of dynamic programming and Monte Carlo methods by updating estimates based on the current reward and the estimated value of the next state. In the simplest form, known as $TD(0)$, the state-value function is updated at each time step according to:

$$V(s_t) \leftarrow V(s_t) + \alpha\big[r_t + \gamma\, V(s_{t+1}) - V(s_t)\big],$$

where the difference $r_t + \gamma V(s_{t+1}) - V(s_t)$ is called the TD error, and α the learning rate. TD updates occur at each step and as a result, the method can learn online and from incomplete episodes (avoiding the high variance), while still converging under appropriate conditions. For the tabular case (where each state has its own value estimate), convergence to the true value function is proven under Robbins-Monro learning rate conditions and sufficient exploration. For on-policy[3] TD(0) with linear function approximation, convergence to the best linear approximation is shown under similar learning rate conditions and exploration [53]. However, for off-policy[4] TD with linear function approximation, convergence is not guaranteed for simple TD but has been proven for algorithms like GTD (Gradient Temporal Difference) and TDC (TD with Correction) under specific conditions [35]. With non-linear function approximation, convergence is generally not guaranteed and can be unstable.

[3]The policy being evaluated is the same policy used to generate the data

[4]The policy being evaluated is different from the policy used to generate the data

SARSA and Q-learning The idea can be extended to action-value learning with two algorithms. SARSA (named for the quintuple: $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$), is an on-policy method that uses the next action actually chosen under the current policy,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_t + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big],$$

thereby learning the value of the policy being executed. Q-learning treats the next-state action as the best possible, regardless of the agent's behavior, so it uses the maximum over all actions at the next state,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_t + \gamma\, \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big].$$

This maximization step guarantees that Q-learning converges to the optimal action-value function under standard assumptions.

Eligibility Traces and TD(λ) TD(λ) merges one-step TD and Monte Carlo by introducing *eligibility traces*, which accumulate credit for past states (or state–action pairs) according to how recently and frequently they have been visited. In the *forward view*, the λ-return is defined:

$$G_t^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)},$$

where $G_t^{(n)}$ is the n-step return. This mixes n-step returns $G_t^{(n)}$, weighting longer returns geometrically by λ. In the equivalent *backward view*, a trace $e_t(s)$ is maintained for each state s and updated by:

$$e_t(s) = \gamma\,\lambda\, e_{t-1}(s) + \mathbb{1}\{s_t = s\},$$

and all states are then updated by the same TD error $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$:

$$V(s) \;\leftarrow\; V(s) \;+\; \alpha\,\delta_t\, e_t(s) \quad \text{for all } s.$$

The weight $\lambda \in [0,1]$ is used to interpolate between pure TD(0) ($\lambda = 0$) and Monte Carlo ($\lambda = 1$), in order to trade off bias and variance in the updates.

Approaches to Solve Continuous MDPs

In continuous Markov decision processes, the infinite cardinality of states or actions disallows exact tabular solutions. Instead, it is necessary to use approximation and gradient–based optimization in order to generalize across similar states and actions.

Recall that the core of solving MDPs is the Bellman expectation equation, which in continuous notation is written as:

$$V^{\pi}(s) \;=\; \int_{\mathcal{A}} \pi(a \mid s) \int_{\mathcal{S}} P(s,a,s')\big[R(s,a,s') + \gamma\, V^{\pi}(s')\big]\, ds'\, da,$$

and its pair, Bellman optimality equation:

$$V^{*}(s) \;=\; \max_{a\in\mathcal{A}} \int_{\mathcal{S}} P(s,a,s')\big[R(s,a,s') + \gamma\, V^{*}(s')\big]\, ds'.$$

These equations define the fixed-point conditions that any candidate value function must satisfy under a given policy π or under the optimal policy π^*. As we explained, in continuous domains we cannot store $V(s)$ for every s, so a parameterized approximator $V_\theta(s)$ is used to search for the parameters θ that push V_θ toward the Bellman target.

In the following paragraphs, we organize the main methods used to solve RL problems described with continuous MDPs.

Key Algorithms for Continuous MDPs

Value-Based Function Approximation When states or actions lie in continuous spaces, it is impossible to store a distinct value or action-value for each state or state–action pair. The standard solution is to utilize a *function approximator*, such as a neural network or a linear combination of basis functions, whose parameters θ are used to define an approximate state-value $V_\theta(s)$ or action-value $Q_\theta(s,a)$. The parameters of these approximators are found by minimizing Bellman error in expectation. For the state-value case we have:

$$\theta^* \;=\; \arg\min_\theta \; \mathbb{E}_{s,a,s'}\Big[\big(V_\theta(s) \;-\; (R(s,a) + \gamma\, V_\theta(s'))\big)^2\Big],$$

where V_θ aims to approximately satisfy the Bellman equation $V(s) = R(s,a) + \gamma\,\mathbb{E}[V(s')]$. In practice the expectation is replaced by samples (s,a,r,s') drawn from a simulated model or real environment trajectories. Analogously, an approximate action-value function $Q_\theta(s,a)$ can be fit by aiming to approximate targets of the form $r + \gamma \max_{a'} Q_\theta(s',a')$.

Policy Optimization Methods An alternative to value-based fitting is to directly parameterize the policy $\pi_\theta(a \mid s)$ and adjust θ to maximize the expected return

$$J(\theta) \;=\; \mathbb{E}_{\tau\sim\pi_\theta}\Big[\sum_{t=0}^{\infty} \gamma^t\, r_t\Big].$$

The policy gradient theorem shows that

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d^\pi,\, a\sim\pi_\theta}\big[\nabla_\theta Q^{\pi_\theta}(s,a)\big],$$

where d^π is the discounted state-visitation distribution. In continuous action spaces, this expectation can be estimated by Monte Carlo and used to perform stochastic gradient ascent on θ.

Deep Deterministic Policy Gradient (DDPG) [39] extends this idea by maintaining both a deterministic policy $\mu_\phi(s)$ and a critic $Q_\theta(s,a)$. The critic is learned off-policy by minimizing the mean-squared Bellman error

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\Big[\big(Q_\theta(s,a) - [\,r + \gamma\, Q_{\theta'}(s', \mu_{\phi'}(s'))\,]\big)^2\Big],$$

where θ', ϕ' are slowly-updated target networks for stability and $\mathcal{D}$ is a replay buffer[5]. The actor parameters ϕ are then adjusted by

$$\nabla_\phi J \approx \mathbb{E}_{s\sim\mathcal{D}}\big[\nabla_a Q_\theta(s,a)\,\big|_{a=\mu_\phi(s)}\;\; \nabla_\phi \mu_\phi(s)\big].$$

[5]The replay buffer is a memory that stores past experiences as tuples of (s,a,r,s') allowing for batched updates and breaking temporal correlations in the data. Learning from this replay buffer is done by classic supervised machine learning algorithms, where given a state s and action a (as input), the goal is to predict a target value (related to the Q-value or policy), which a task that can be solved effectively.

Proximal Policy Optimization (PPO) [40] improves stability of on-policy policy gradient by optimizing a clipped surrogate objective. Defining the probability-ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ and an advantage estimate $\hat{A}_t$, PPO maximizes

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r_t(\theta)\,\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\,\hat{A}_t\big)\Big],$$

thereby preventing any single policy update from moving $r_t(\theta)$ outside $[1-\epsilon, 1+\epsilon]$.

Actor–Critic Methods *Actor–Critic* (A2C) algorithms blend value-based and policy-gradient ideas. A "critic" network Q_θ or V_θ evaluates the current policy's performance, while an "actor" network π_ϕ is updated in the direction suggested by the critic. Both on-policy (e.g. A2C) and off-policy (e.g. DDPG, SAC) variants exist. The critic's bootstrapped estimates reduce the variance of the policy gradient, while the actor's direct optimization steers behavior toward higher return.

3.6.4 Model-Free vs. Model-Based Reinforcement Learning

RL approaches can be categorized into model-free and model-based, distinguished by whether or not the agent learns or has a model of the environment's dynamics. A model of the environment gives the agent the ability to predict the next state and the immediate reward given the current state and action. Model-based RL requires learning this transition function $P(s' \mid s, a)$ and the reward function $R(s, a)$ or $R(s, a, s')$. Once a (sufficiently accurate) model exists, the agent can use it to plan sequences of actions by simulating future trajectories. The advantage that model-based RL has is that by learning the underlying dynamics, the agent can often achieve good performance with fewer real-world interactions. However, the challenge is to learn an accurate model, especially in complex or stochastic environments.

In contrast, model-free RL methods do not attempt to explicitly learn the environment's model. Instead, they directly learn the value function $V(s)$ or $Q(s, a)$ or the policy $\pi(a|s)$ from experience. The algorithms we analyzed previously (Monte Carlo methods, Temporal Difference learning, SARSA and Q-learning, and policy gradient methods) refer to model-free RL. Despite the simplicity of implementation, due to it not requiring to learn a model of the environment, model-free RL can be surprisingly effective in various problems. However, these methods often require a large number of interactions with the environment to learn optimal policies, as they must directly experience the consequences of their actions, in order to understand the underlying dynamics.

The choice between model-free and model-based RL depends on the complexity of the environment, the availability of a good model, and the amount of interaction data that can be collected.

3.6.5 Practical Considerations in Reinforcement Learning

Although in theory, the foundations of RL provide a strong basis for developing intelligent agents, the actual training of these agents has many practical obstacles. Effective learning and stable behavior requires carefully choosing several aspects of the learning process.

Reward Design

The reward signal is fundamental in the learning process, being the primary feedback mechanism that guides the agent to learn effectively. A well-designed reward function should be relevant to the desired task, in order to provide signals that encourage the agent to learn how to achieve its goals. However, reward design can be a difficult task.

One example is using sparse rewards. In settings where the agent receives a non-zero reward only upon achieving a final goal, the learning process can be extremely challenging, especially in complex environments with long horizons. The reason is that the agent may struggle to associate its actions with the eventual success.

Conversely, dense rewards, provide feedback at each step, thus can accelerate learning. However, the trade-off is that this type of reward may inevitably guide the agent towards suboptimal behaviors if it is not carefully designed. Various techniques (reward shaping for instance) can be employed to mitigate this problem, but they carry the risk of introducing undesired local optima that the agent might converge to. Furthermore, the scale of the reward has a significant role in the learning stability and speed.

Function Approximation

As previously discussed, many real-world problems involve continuous or very large discrete state and action spaces, thus handling tabular representations of value functions or policies is infeasible. Function approximation techniques are essential to generalize from experienced states and actions to unseen ones. The choice of function approximator, can be a linear combination of basis functions, a neural network, or another differentiable function, and it impacts the learning process and the agent's ability to represent complex relationships. The function approximator should not only be able to capture the underlying structure of the value function or policy, but also to avoid overfitting to the training data. Moreover, the stability of learning can be affected by the choice of function approximator.

Exploration Strategies

The exploration-exploitation dilemma is a central challenge in reinforcement learning, and the strategy that balances them affects the learning efficiency and the quality. As mentioned earlier, insufficient exploration leads to the agent converging to a suboptimal policy, while excessive exploration results in slow learning and suboptimal behavior during the training.

Many exploration strategies have been developed to address this challenge. Simple approaches like ϵ-greedy exploration [36], where the agent takes a random action with probability ϵ and the greedy action with probability $1 - \epsilon$, are used to balance the exploration and exploitation in a simple and effective way. More sophisticated methods, such as *Upper Confidence Bound* (UCB) [36] algorithms, and *Thompson Sampling* [54], which is a probabilistic strategy where the agent maintains a belief (probability distribution) over the value of each action and samples from these distributions to select the next action. Actions with higher sampled values are more likely to be chosen, thus implicitly balancing exploration of uncertain but potentially good actions. In continuous action spaces, exploration often involves adding noise to the chosen action or exploring in the parameter space of the policy.

Initialization

The initial values of the function approximator's parameters can have a significant impact on the learning process. Poor initialization leads to slow convergence, instability, or even converging in suboptimal regions of the parameter space.

Normalization

Scaling and shifting the input and sometimes output to a standard range (e.g., between -1 and 1, or having zero mean and unit variance) is common practice in machine learning that can also benefit reinforcement learning. Normalization can improve the stability and speed of learning, especially in gradient-based optimization with function approximators. Features with different scales can lead to gradients that are also on different scales, causing oscillations and slow convergence. By normalizing the state space, its features contribute more equally to the learning process. Also, normalizing the action space results in the output of the policy network to be within a suitable range for the environment and can stabilize the training.

Stability of Learning

The process of training, especially when using complex function approximators (such as deep neural networks), can be inherently unstable. Several factors lead to this, such as the non-stationary nature of the target functions (the policy and value functions are constantly updated), correlated updates, and issues with gradient propagation. To mitigate instabilities, various techniques are used. Experience replay allows the agent to learn from past experiences by storing transitions in a buffer and sampling them randomly, reducing the correlation between consecutive updates. Target networks use a delayed copy of the value function or policy network to compute target values. This stabilizes learning by reducing the feedback loop between the updating network and the target (the online network's weights are updated after each training step to improve its predictions, so using these rapidly changing weights to also define the target values would create a moving and unstable learning objective). Gradient clipping limits the magnitude of gradients during backpropagation, which could destabilize the learning process.

3.7 Model Predictive Control

Introduction to Model Predictive Control

Model Predictive Control (MPC) is a control strategy that relies on a dynamic model of the system to predict its future evolution over a finite horizon. At each time step, MPC solves an open-loop[6] optimal control problem, determining a sequence of future control actions that optimizes a predefined objective function while satisfying the system's dynamics and constraints. A feature of MPC is the receding horizon principle: only the first control action of the calculated optimal sequence is implemented, and the optimization process is repeated at the next time step, where the updated system state is the new initial condition. This iterative optimization helps in managing disturbances, while following the system's dynamics and constraints.

Core Components of Model Predictive Control

The general MPC version revolves around three concepts: *system modeling*, *prediction*, and *optimization* with receding horizon.

System Modeling MPC uses an explicit model of the system to predict its future behavior. The MPC framework works both with discrete-time and continuous-time systems.

For a *discrete-time system*, the dynamics are usually represented in state-space form:

$$x(k+1) = f(x(k), u(k), d(k)),$$

where $x(k)$ is the state vector, $u(k)$ is the control input vector, and $d(k)$ denotes disturbances in the system. In the case of a linear time-invariant (LTI) system, the system takes the form:

$$x(k+1) = Ax(k) + Bu(k) + Ed(k),$$

where A, B, E are system matrices.

For a *continuous-time* system, the dynamics are described by differential equations:

$$\dot{x}(t) = f(x(t), u(t), d(t)),$$

and in the LTI case:

$$\dot{x}(t) = Ax(t) + Bu(t) + Ed(t).$$

Prediction Using the model and the current state $x(t)$, MPC predicts the future trajectory of the system over a finite horizon N (T in continuous time) for a sequence of future control actions $U = [u(t), u(t+1), \ldots, u(t+N-1)]$ ($u(\tau)$ for $\tau \in [t, t+T)$ in continuous time). The predicted state and output trajectories are denoted $x(k|t)$, $y(k|t)$ for discrete time, and $x(\tau|t)$, $y(\tau|t)$ for continuous time, based on information available at time t. The control horizon N (or T) defines how many future control moves the algorithm considers as decision variables to optimize.

[6] Open-loop control computes a control sequence based on a model and initial state, without feedback during execution.

Optimization with Receding Horizon At each time step t, MPC solves an optimal control problem to find the control sequence U that optimizes a functional J over the prediction horizon, subject to the system dynamics.

For instance, for the discrete-time case, a generic cost functional would be:

$$J_t\big(x(\cdot|t), u(\cdot|t)\big) = \sum_{k=t}^{t+N-1} \ell\big(x(k|t), u(k|t)\big) + V_f\big(x(t+N|t)\big)\,,$$

where $\ell(x, u)$ is a stage cost and $V_f(\,\cdot\,)$ is a (optional) terminal cost. The receding-horizon optimization at time t is:

$$\begin{aligned} \underbrace{\text{minimize}}_{u(t|t),\ldots,u(t+N-1|t)} \quad & J_t\big(x(\cdot|t), u(\cdot|t)\big) \\ \text{subject to} \quad & x(k+1|t) = f\big(x(k|t), u(k|t)\big) \quad \forall\, k = t, \ldots, t+N-1, \\ & x(k|t) \in \mathcal{X},\ u(k|t) \in \mathcal{U} \quad \forall\, k = t, \ldots, t+N-1, \\ & x(t|t) = x_t \quad \text{(current state)}. \end{aligned}$$

Here f denotes the system dynamics, and $\mathcal{X}, \mathcal{U}$ are state/input constraint sets. Once the sequence $\{u^*(k|t)\}$ is calculated by the optimizer, only $u^*(t|t)$ is applied, and at the next time step $t+1$ the state is observed again, and the entire finite-horizon problem is solved again over $[t+1, t+N]$.

This receding-horizon scheme provides stability since it allows replanning at every step to account for disturbances and modeling errors.

Solving the Optimization Problem

The nature of the system model (linear or not), the optimization objective and the constraints determine the type of optimization problem that needs to be solved at each MPC step.

For instance, the optimization problem for linear systems with quadratic objective functions and linear constraints, can be formulated as a Quadratic Program (QP). QPs' property is that they have a unique global optimum if a solution exists, and can be solved efficiently with well-known numerical algorithms. Thus linear MPC is suitable for real-time applications.

For nonlinear systems, objective function or constraints, the optimization problem becomes a Nonlinear Program (NLP). NLPs are usually more challenging to solve. They may have multiple local optima, and finding one is not guaranteed. Iterative methods are the common solution approaches. The computational complexity can thus limit the applicability of nonlinear MPC. Real-time iteration (RTI) schemes have been developed to address these challenges by performing only a few iterations, in order to find a feasible and sufficiently good solution.

Considering continuous-time MPC, the optimization aims to find an optimal control trajectory over a time interval. This can be approached using methods from optimal control theory, such as direct methods (e.g., discretization of the control and state trajectories and then solving a finite-dimensional optimization problem) or indirect methods (e.g., using Pontryagin's maximum principle to derive conditions for optimality). Discretization often leads to large-scale optimization problems which can be solved using NLP solvers.

Applications of Model Predictive Control

In many engineering domains (chemical plants, robots, vehicles, power grids, etc) Model Predictive Control is used for its ability to anticipate future behavior, while respecting constraints, and

handle multiple interacting variables in a single optimization. In process plants, MPC adjusts flows, temperatures and compositions. In robotics and autonomous vehicles, it plans motion by predicting kinematics and dynamics ahead, resulting in smooth control and obstacle avoidance. Applications such as adaptive cruise control, lane-keeping and emergency braking also use MPC to ensure the safety on the road (constraint) while optimizing aspects like fuel consumption and trip duration. Electric power systems use MPC to balance supply and demand, regulate voltage, and integrate renewables which are characterized by rapid fluctuations.

Chapter 4

Methodology

Contents

4.1 Motivation

The integration of advanced machine learning models with optimization and control methods is becoming increasingly popular in modern decision-making systems. In many real-world applications, sequential decisions must be made under uncertainty, based on sequential observations, where the future (model of the system) is not fully known, but predictions can be made with various degrees of confidence. Model-Based Reinforcement Learning techniques have been utilized to solve such problems by assuming a model of the world's system and making decisions based on it, while adapting on the observed data.

As discussed in previous sections, diffusion models were initially developed for applications regarding image generation, with remarkable capabilities in learning complex probability distributions, thanks to their inherent complexity. Thus, they are suitable to model the probability distribution of the future data points of multivariate time series. These probabilistic forecasts incorporate the uncertainty of the future which is critical in the task of controlling real-world stochastic systems that depend on unknown or unobservable parameters. Examples of such systems, where it is impossible (or really hard) to measure all the parameters that affect the future, are the *Weather*, which depends on atmospheric conditions that are influenced by chaotic factors, some *Quantum Systems*, where it is impossible to measure accurately all the parameters, *Human Behavior*, which is influenced by emotions, social interactions and cognitive biases that are not measurable and *Financial Markets*, that are both chaotic and depend on politics and human psychology.

This thesis investigates the synergy between diffusion models for probabilistic time series forecasting and sequential decision-making frameworks like Model Predictive Control (MPC), focusing on their combined ability to enhance optimal decision-making processes. We aim to overcome the challenges caused by the high-dimensional and uncertain dynamics, by using the predictions of diffusion models as a stochastic model of the system we try to control, leading to more informed and robust decisions.

4.2 Problem Definition

In most real-world decision-making problems, the true state of the environment, which is all the information an agent would need in order to make optimal decisions, is not directly measurable. Instead, an agent must use noisy or partial observations as a surrogate to the real state. To propose our method, we formally model the class of problems that we aim to solve as a discrete-time Partially Observable Markov Decision Process (POMDP). A POMDP is defined by the 7-tuple

$$(\mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma),$$

where:

- $\mathcal{S}$ is the (hidden) state space, representing all possible configurations of the environment.
- $\mathcal{A}$ is the set of actions available to the agent.
- $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state transition function; for any current state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$, $T(s' \mid s, a)$ gives the probability of transitioning to state s'.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which assigns an immediate reward $R(s, a)$ based on the (unobserved) state s and the chosen action a. In practice, since the true state is not observable, the reward is computed via a belief function based on the observations.
- Ω is the set of all possible observations.
- $O : \mathcal{S} \times \mathcal{A} \to \Delta(\Omega)$ is the observation function; when the environment transitions to a new state s' after an action a is taken in state s, the agent receives an observation $o \in \Omega$ with probability $O(o \mid s', a)$. Note that the observation o may capture both deterministic and stochastic aspects of the state.
- $\gamma \in [0, 1)$ is the discount factor, which governs the trade-off between immediate and future rewards.

At each time step t, the environment is in some (hidden) state $s_t \in \mathcal{S}$. The agent does not observe s_t directly; instead, it receives an observation $o_t \in \Omega$ that provides partial information about the state. Based on this observation, the agent selects an action $a_t \in \mathcal{A}$, which results in the environment transitioning to a new state s_{t+1} according to $T(s_{t+1} \mid s_t, a_t)$. Simultaneously, the agent obtains a new observation $o_{t+1} \sim O(\cdot \mid s_{t+1}, a_t)$ and an immediate reward $r_t = R(o_t, a_t)$.

Moreover, the proposed method refers to problems where the system is not fully observable and where the dynamics are partially deterministic and partially stochastic. At each time step t (or with some delay), the agent receives an observation $o_t \in \Omega$ that provides partial information about the hidden state. In our formulation, we assume that the observation o_t can be decomposed into two components:

$$o_t = \langle o_t^d,\, o_t^s \rangle,$$

where o_t^d is the *deterministic* component of the observation, which is directly affected by the agent's actions. For example, in a battery storage application, the state-of-charge (SoC) evolves deterministically according to the applied control inputs. The o_t^s is the *stochastic* component that captures the uncertainty of the system (e.g., energy prices that follow complex, stochastic dynamics).

To plan its actions, the agent forecasts the trajectory of future observations, regarding their stochastic component. In particular, the agent generates forecasts of the form $\hat{o}^s_{t+1}, \hat{o}^s_{t+2}, \ldots$, and calculates the optimal action sequence $a_{t+1}, a_{t+2}, \ldots$, that will deterministically lead to observations $\hat{o}^d_{t+1}, \hat{o}^d_{t+2}, \ldots$, completing the observation trajectory $\hat{o}_{t+1}, \hat{o}_{t+2}, \ldots$. This trajectory helps calculate the expected cumulative reward, based on the model of the system, that the agent aims to maximize, through the action sequence.

In this formulation, we use diffusion-based time series forecasting models, such as TimeGrad, to predict the evolution of the stochastic part of the observations, providing the agent with an uncertainty-aware and powerful model that will be integrated into control algorithms like Model Predictive Control (MPC).

4.3 Probabilistic Forecasting for POMDPs

In our problem setup, to effectively plan actions under uncertainty, the agent has to estimate the evolution of the stochastic component of the observations. In our approach, a diffusion-based time series forecasting model, TimeGrad, provides a probabilistic model of the stochastic dynamics.

TimeGrad is designed to learn the conditional probability distribution of future stochastic observation components given historical data. Let $\mathbf{o}_k^s$ denote the stochastic part of the observation at time k. Given a history of stochastic observations $\mathbf{o}_{1:k_0-1}^s$, TimeGrad approximates the distribution of future values over a prediction horizon $k = k_0, \ldots, N$ as follows:

$$q\left(o_{k_0:N}^s \mid o_{1:k_0-1}^s\right) = \prod_{k=k_0}^{N} q\left(o_k^s \mid o_{1:k-1}^s\right) \approx \prod_{k=k_0}^{N} p_\theta(o_k^s|\mathbf{h}_{k-1}), \quad \mathbf{h}_k = \mathrm{RNN}_\theta(o_k^s, \mathbf{h}_{t-1}),$$

where p_θ is the learned forecast distribution (θ denotes the learned parameters), and $\mathbf{h}_{1:k_0-1}$ is the hidden RNN state that encodes the important information from the past observations.

Within our POMDP formulation, the agent uses these probabilistic forecasts to guide its control strategy. While the deterministic component of the observation (e.g., battery state-of-charge) is governed by known dynamics, the stochastic component is estimated using TimeGrad. The agent combines the known deterministic dynamics, o_k^d, with the forecasted stochastic ones, $\hat{o}_k^s$, to construct predicted observations:

$$\hat{o}_k = \langle o_k^d, \hat{o}_k^s \rangle.$$

This forecasted observation sequence is then integrated into control algorithms, such as Model Predictive Control (MPC) and its stochastic variants, to determine an optimal action sequence that maximizes the expected cumulative reward.

4.4 Model Predictive Control with TimeGrad

Our first step, is to integrate the diffusion-based forecasting model into a Deterministic Model Predictive Control framework to calculate the optimal control sequence in the partially observable environment.

At each decision epoch, given a history of observed stochastic components $o_0^s, o_1^s, \ldots, o_k^s$, TimeGrad generates a distribution of M forecasts $\{\hat{o}_{k+1}^{s,(i)}\}_{i=1}^{M}$ for the next time step. We aggregate these samples using a statistical operator, such as the median or the mean, in order to yield a single point forecast:

$$\hat{o}_{k+1}^s = \mathcal{F}(o_0^s, \ldots, o_k^s) = \text{median}\Big\{\hat{o}_{k+1}^{s,(i)}\Big\}_{i=1}^{M}.$$

The overall next-step observation $\hat{o}_{k+1} = \langle o_{k+1}^d, \hat{o}_{k+1}^s \rangle$, is constructed by calculating the deterministic part o_{k+1}^d as well, after choosing an action a_k.

Using this forecast operator in an autoregressive manner, we generate a predicted observation trajectory for a future horizon of N steps. The deterministic MPC problem, when starting at time step k_0 and planning for N steps ahead, is formulated as follows:

$$\begin{aligned}
\underset{a_{k_0},\ldots,a_{N+k_0-1}}{\text{maximize}} \quad & \sum_{k=k_0}^{N+k_0-1} R\big(\hat{o}_k, a_k\big) \\
\text{subject to} \quad & o_0, \ldots, o_{k_0-1} \quad \text{are observed}, \\
& \hat{o}_{k+1} = \mathcal{F}(o_0, \ldots, o_k), \quad \text{for } k = k_0 - 1, \ldots, N + k_0 - 2, \\
& a_k \in \mathcal{A}(s_k), \quad \text{for } k = k_0, \ldots, N + k_0 - 1.
\end{aligned}$$

Here, $R(\hat{o}_k, a_k)$ denotes the reward obtained when applying action a_k based on the predicted observation $\hat{o}_k$. Note that by aggregating multiple probabilistic forecasts into one (via the median or mean), the approach handles uncertainty solely at the forecast level. Consequently, the MPC optimization problem becomes deterministic, as it is based on a single predicted trajectory of future observations.

The following step is to extend this framework to fully account for uncertainty within the optimization process (using stochastic MPC variants).

4.5 Stochastic Model Predictive Control with TimeGrad

To account for the uncertainty in the evolution of the unobserved stochastic component of the environment during the optimization process, we extend the classical MPC framework by integrating multiple forecast trajectories. In this formulation, the distribution of future stochastic observations is explicitly incorporated into the optimization problem. We present two variants of Stochastic MPC (SMPC): one using Monte Carlo simulations and the other using predictions organized in a tree structure.

4.5.1 SMPC with Monte Carlo Simulations

In Monte Carlo SMPC, we model the uncertainty by sampling M independent trajectories from TimeGrad's forecast distribution. For each scenario $i \in \{1, \dots, M\}$, the future stochastic component is generated recursively over the prediction horizon $k = k_0, \dots, N + k_0 - 1$ as

$$o_{k+1}^{s,(i)} \sim p_\theta\Big(o_{k+1}^{s} \mid \mathbf{h}_k\Big).$$

For each scenario i, the full predicted observation at time $k+1$ is

$$\hat{o}_{k+1}^{(i)} = \langle o_{k+1}^{d},\, o_{k+1}^{s,(i)} \rangle,$$

where o_{k+1}^{d} is computed deterministically from the known system dynamics. The cumulative reward along the ith trajectory, based on the actions applied is defined as

$$\mathcal{J}^{(i)} = \sum_{k=k_0}^{N+k_0-1} R\big(\hat{o}_k^{(i)},\, a_k\big), \quad \text{where } \hat{o}_k^{(i)} = \langle o_k^{d},\, \hat{o}_k^{s,(i)} \rangle$$

The corresponding SMPC optimization problem seeks the action sequence $\{a_{k_0}, \dots, a_{N+k_0-1}\}$ that maximizes the average cumulative reward across all M scenarios:

$$\begin{aligned}
\underset{a_{k_0},\dots,a_{N+k_0-1}}{\text{maximize}} \quad & \frac{1}{M} \sum_{i=1}^{M} \mathcal{J}^{(i)} \\
\text{subject to} \quad & o_0, \dots, o_{k_0-1} \text{ are observed}, \\
& o_{k+1}^{s,(i)} \sim p_\theta\Big(o_{k+1}^{s} \mid \mathbf{h}_k\Big), \quad k = k_0, \dots, N + k_0 - 1,\ \forall i, \\
& a_k \in \mathcal{A}, \quad k = k_0, \dots, N + k_0 - 1.
\end{aligned}$$

This formulation integrates uncertainty over the entire prediction horizon by considering multiple trajectory realizations, leading to robust control decisions, accounting for the stochastic dynamics in the optimization process.

4.5.2 Scenario Tree-Based MPC

Multistage stochastic programming seeks an optimal policy over a finite horizon by discretizing the future uncertainty into a scenario tree. A scenario tree has for root the last realized random variable of a stochastic process and nodes at each stage t which represent possible realizations of the random process up to t. Moreover, the tree branches carry probabilities consistent with an

assumed distribution of the stochastic process. It is important to mention that the tree respects the *non-anticipativity* constraints of the stochastic process, by requiring that decisions made at stage t can depend only on the stochastic variables realized up to t. By construction, a proper scenario tree has the following constraints: all scenarios that share the same history up to stage t coalesce at the same node, so the decision at that node is the same across those scenarios. We propose two Scenario Tree implementations.

A. Forward Clustering Scenario Tree Construction

In the forward clustering variant, we grow the scenario tree one stage at a time, starting from the current state and extending into the future. At each node, we call the diffusion model to produce a set of M forecasts covering the next stage. We then apply a standard K-means clustering to those M trajectories, and treat each cluster's centroid as the representative of the branch. The centroid may be chosen as the arithmetic mean, component-wise median, medoid, or using any other aggregation operator. The proportion of samples falling into each cluster becomes the node probability, and each node's cumulative probability is the product of probabilities along the path from the root. In order to avoid the exponential expansion of the tree, one should trim it online, for example by maintaining the L highest-probability children at each level, preserving the most likely scenarios. This forward scheme is straightforward, and it naturally enforces non-anticipativity as well, since each node's descendants are generated only from the information available up to that node. The forward-clustering procedure is defined in Algorithm 1.

B. Backward-Hierarchical Clustering Scenario Tree Construction

The backward hierarchical variant first samples a full set of M trajectories spanning the entire decision horizon (consisting of all stages) in one shot. It then builds the tree, starting from the leaves and ending on the root, by repeatedly clustering those full-length samples at decreasing temporal resolutions. At the deepest level (end of the stage horizon), the M sampled trajectories are grouped into K_D (a pre-defined parameter) number of clusters to form the last-stage nodes. The cluster centroid represents the node's grouped forecasts, while the branch probability is defined again as the proportion of samples falling into the group. Next, to form the previous stage, we merge each leaf-cluster's trajectories and recluster them into K_{D-1} groups—ensuring clusters (and thus decisions) only ever coalesce, never split, maintaining non-anticipativity. This merging-then-reclustering continues until reaching the root. A Lloyd-style loop optimizes a clustering objective that balances reconstruction error, within-cluster variance, and between-cluster separation. In this way, the algorithm chooses both cluster assignments and the number of clusters without the need of setting arbitrary thresholds. The backward-hierarchical clustering method is defined in Algorithm 2.

Algorithm 1: Forward Clustering Scenario Tree Construction

Input:

$\mathcal{S}$: empty tree with root;
$\mathcal{O}$: initial observation history up to current time;
predictor: trained diffusion-based forecaster (e.g. TimeGrad);
D: tree depth (number of stages); K: clusters per stage; M: forecast samples;
H: horizon per stage; F: feature dimension; L: top-L children per node.

Output:

Populated scenario tree $\mathcal{S}$ with forecasts, probabilities, and representative paths.

Initialize queue $Q \leftarrow \{(\mathcal{S} = \text{root}, \mathcal{O}, 0)\}$, node_counter $= 0$;
while $Q \neq \emptyset$ **do**
  $(node, \mathcal{O}, \ell) \leftarrow \text{Dequeue}(Q)$;
  `// Assign unique identifier to node`
  $node$.id $\leftarrow$ node_counter; node_counter++;
  `// If desired depth reached then stop expanding the current path`
  if $\ell \geq D$ **then**
    continue
  `// Generate M stochastic forecasts`

$$\{\mathcal{F}^{(i)}\}_{i=1}^{M} \;=\; \text{predictor.sample}(\mathcal{O}, M), \quad \mathcal{F}^{(i)} \in \mathbb{R}^{H \times F}$$

  Reshape forecasts into $X \in \mathbb{R}^{M \times (H \cdot F)}$;
  Cluster forecasts with K-means $\rightarrow$ cluster labels $\{\ell_i\}$;
  `// Compute representative forecast and probabilities`
  for $k = 1, \ldots, K$ **do**
    $\mu_k \leftarrow$ centroid of cluster k;
    $p_k \leftarrow \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}(\ell_i = k)$;
    $P_k \leftarrow node.P \cdot p_k$;
    Create child node v_k with forecast μ_k, probability P_k;
  `// Select top-`L` children based on cumulative probability`
  Sort $\{v_k\}$ by P_k in descending order and keep top L;
  `// Expand the tree`
  for *each kept child-node* v_k *with representative* μ_k **do**
    Concatenate $\mathcal{O}' \leftarrow \mathcal{O} \,\|\, \mu_k$;
    Attach v_k as child of $node$;
    Update tree $\mathcal{S}$ with v_k;
    Enqueue$(Q, (v_k, \mathcal{O}', \ell + 1))$;
return $\mathcal{S}$

Algorithm 2: Backward-Hierarchical Clustering Scenario Tree Construction (Phase 0)

Input:

$\mathcal{S}$: empty tree with root;
$\mathcal{O}$: initial observation history;
`predictor`: pretrained diffusion-based forecaster;
D: number of tree levels (depth);
M: number of trajectories to sample;
N: forecasting horizon (number of time-steps per trajectory);
$\{K_d\}_{d=1}^{D}$: desired number of clusters at each level d (can be calculated automatically);
$\{w_1, w_2, w_3\}$: weights for the clustering objective.

Output:

A fully-constructed scenario tree $\mathcal{S}$ with levels $1, \ldots, D$ and branch probabilities.

// Definition of clustering-objective terms (for any clustering $C = \{C_1, \ldots, C_K\}$ on data $X = \{\mathbf{x}_i\}_{i=1}^{M}$):

1. Reconstruction Error:

$$RE \;=\; \sum_{i=1}^{M} \big\|\mathbf{x}_i \;-\; \mu_{c(i)}\big\|^2, \quad RE_0 \;=\; \sum_{i=1}^{M} \big\|\mathbf{x}_i \;-\; \bar{\mathbf{x}}\big\|^2,$$

2. Intra-Cluster Variance:

$$IntraVar \;=\; \sum_{k=1}^{K} \sum_{\mathbf{x}\in C_k} \mathrm{Var}(\mathbf{x}), \quad IntraVar_0 \;=\; \sum_{j=1}^{F} \mathrm{Var}\big(X_{:,j}\big) \;\times K,$$

3. Inter-Cluster Separation:

$$InterSep \;=\; \sum_{1\le k<\ell\le K} \big\|\mu_k \;-\; \mu_\ell\big\|^2, \quad InterSep_0 \;=\; \sum_{1\le k<\ell\le K} \big\|r_k \;-\; r_\ell\big\|^2,$$

where: $\mathbf{x}_i \in \mathbb{R}^{N\cdot F}$ is the ith flattened trajectory, F is the feature-dimensionality per time-step, μ_k is the centroid of cluster C_k, $c(i)$ denotes the cluster-index of $\mathbf{x}_i$, $\bar{\mathbf{x}}$ is the global mean of all $\{\mathbf{x}_i\}$, $r_1, \ldots, r_K$ are K random initial centers used to compute $InterSep_0$, $RE_{\text{norm}} = RE/(RE_0 + \varepsilon)$, $IntraVar_{\text{norm}} = IntraVar/(IntraVar_0 + \varepsilon)$, $InterSep_{\text{norm}} = InterSep/(InterSep_0 + \varepsilon)$, and the composite objective is

$$\text{Obj} \;=\; w_1\, RE_{\text{norm}} \;+\; w_2\, IntraVar_{\text{norm}} \;-\; w_3\, InterSep_{\text{norm}}.$$

// Note: $RE_0, IntraVar_0, InterSep_0$ are baseline scores computed once on the unclustered data (by the overall variance and random initial centers), used to normalize each metric-ensuring RE, IntraVar, and InterSep become scale-invariant and comparable across the clusterings.

// Phase 0: Sample all M full-horizon trajectories from the diffusion-based predictor

$$\{\mathbf{x}^{(i)}\}_{i=1}^{M} \;=\; \texttt{predictor.sample}\big(\mathcal{O},\, M\big) \quad \text{with each } \mathbf{x}^{(i)} \in \mathbb{R}^{N\times F}.$$

Flatten each $\mathbf{x}^{(i)}$ into $\tilde{\mathbf{x}}^{(i)} \in \mathbb{R}^{N\cdot F}$. Denote the resulting matrix by

$$X \;=\; \big[\,\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(M)}\big]^{\top} \;\in\; \mathbb{R}^{M\times(N\cdot F)}.$$

Algorithm 2: Backward-Hierarchical Clustering Scenario Tree Construction (Phase 1)

`// Define custom_kmeans algorithm, which is a Lloyd-style k-means loop that clusters the data to optimize the composite objective Obj`

`// Phase 1: Cluster at the deepest stage` $d = D$ `(i.e., last time-slice` $t = N$`)`

1. **Compute**

$$\big\{\mu_k^{(D)}\big\}_{k=1}^{K_D},\ \big\{\,\ell_i^{(D)}\big\}_{i=1}^{M} \;=\; \texttt{custom\_kmeans}\big(X,\ K_D,\ w_1, w_2, w_3\big),$$

where $\ell_i^{(D)} \in \{1, \ldots, K_D\}$ is the cluster-label of $\tilde{\mathbf{x}}^{(i)}$ at level D, and $\mu_k^{(D)} \in \mathbb{R}^{N \cdot F}$ is the centroid of cluster k.

2. **For each** $k = 1, \ldots, K_D$:

- Let $S_k^{(D)} = \{\, \mathbf{x}^{(i)} : \ell_i^{(D)} = k \,\}$ be the set of original (unflattened) trajectories in cluster k.
- Branch-probability:

$$p_k^{(D)} \;=\; \frac{|S_k^{(D)}|}{M}.$$

- Representative trajectory (node prediction):

$$\bar{\mathbf{y}}_k^{(D)} \;=\; \text{reshape}\big(\mu_k^{(D)},\, N,\, F\big).$$

- Create a new Stage-D node $v_k^{(D)}$ with:

$$\begin{aligned} v_k^{(D)}.\text{own\_pred} &= \bar{\mathbf{y}}_k^{(D)}, \\ v_k^{(D)}.\text{probability} &= p_k^{(D)}, \\ v_k^{(D)}.\text{trajectories} &= S_k^{(D)}. \end{aligned}$$

- Attach $v_k^{(D)}$ as a child of the unique Stage-$(D-1)$ "parent placeholder" (for now, all children are temporarily orphans under a dummy root).

Algorithm 2: Backward-Hierarchical Clustering Scenario Tree Construction (Phase 2)

`// merge_clusters groups the prior-stage clusters into` j `hyper-groups, evaluating the objective Obj on each merged set and selecting the grouping that minimizes Obj.`

`// Phase 2: For each stage` $d = D-1, D-2, \ldots, 1$`, perform backward merging into` K_d `clusters`

for $d \leftarrow D-1$ **to** 1 ***by*** -1 **do**

1. Collect the list of existing Stage-$(d+1)$ clusters' trajectory-sets:

$$\mathcal{C}^{(d+1)} \;=\; \big\{\, S_k^{(d+1)} \big\}_{k=1}^{K_{d+1}}, \quad \text{where } S_k^{(d+1)} \subset \{\mathbf{x}^{(i)}\}_{i=1}^{M}.$$

2. **for** $k = 1, \ldots, K_d$ **do**

   - **Apply** candidate merge of the K_{d+1} prior clusters into k groups:

$$\{\mu_j^{(d)},\, \ell_i^{(d)}\} \;=\; \texttt{merge\_clusters}\big(\mathcal{C}^{(d+1)},\, k,\, w_1, w_2, w_3\big).$$

   - $\mu_j^{(d)} \in \mathbb{R}^{N\times F}$: centroid of merged group j.
   - $\ell_i^{(d)} \in \{1, \ldots, k\}$: label of each prior group $S_i^{(d+1)}$.
   - **Compute** the composite objective $\mathrm{Obj}(k)$ on these k merged groups.

3. **Select** the best number of clusters

$$k^* \;=\; \arg\min_{1\le k\le K_d} \mathrm{Obj}(k),$$

   and let $\{\mu_j^{(d)},\, \ell_j^{(d)}\}_{j=1}^{k^*}$ be the centroids and labels returned for k^* clusters.

4. **Using** the chosen $\{\mu_j^{(d)}, \ell_j^{(d)}\}$, for each new cluster $j = 1, \ldots, k^*$:

   - Merge trajectories:

$$G_j^{(d)} = \bigcup_{\substack{k=1 \\ \ell_k^{(d)}=j}}^{K_{d+1}} S_k^{(d+1)}.$$

   - Compute branch-probability:

$$p_j^{(d)} = \frac{|G_j^{(d)}|}{M}.$$

   - Extract representative prediction at time $t = d$:

$$\tilde{\mu}_j^{(d)} = \big[\,\mathrm{reshape}(\mu_j^{(d)}, N, F)\big]$$

   - Create Stage–d node $v_j^{(d)}$ with

$$v_j^{(d)}.\mathrm{own\_pred} = \tilde{\mu}_j^{(d)}, \quad v_j^{(d)}.\mathrm{probability} = p_j^{(d)}, \quad v_j^{(d)}.\mathrm{trajectories} = G_j^{(d)}.$$

   - Attach each prior node $v_k^{(d+1)}$ for which $\ell_k^{(d)} = j$ as a child of $v_j^{(d)}$.

Algorithm 2: Backward-Hierarchical Clustering Scenario Tree Construction (Phase 3)

`// Phase 3: Finalize the root (`$d = 0$`)`

- Create the root node $v^{(0)}$ with:

$$v^{(0)}.\text{own\_pred} = \varnothing, \quad v^{(0)}.\text{probability} = 1, \quad v^{(0)}.\text{trajectories} = \{\mathbf{x}^{(i)}\}_{i=1}^{M}.$$

- Attach each Stage-1 node $v_j^{(1)}$ (for $j = 1, \ldots, K_1$) as a child of $v^{(0)}$.

return $\mathcal{S}$

Solving the Optimization Problem

In both variants, once the tree $\mathcal{T}$ is built we solve the following multistage SMPC:

$$\begin{aligned}
\underset{\{a_t^i\}}{\text{maximize}} \quad & \sum_{k=k_0}^{k_0+N-1} \sum_{i \in \mathcal{T}_k} \pi_i \; R\big(\hat{o}_t^i,\; a_t^i\big) \\
\text{subject to} \quad & \hat{o}_{k_0}^1 = o_{k_0}, \\
& \hat{o}_{k+1}^{i,d} \;=\; f\big(\hat{o}_k^{\text{pre}(i),d},\; a_k^{\text{pre}(i)}\big) \quad \forall\, i \in \mathcal{T}_{k+1} \setminus \{1\}, \\
& a_k^i = a_k^j \quad \text{whenever nodes } i, j \text{ share the same history up to } k, \\
& a_k^i \in \mathcal{A} \quad \forall\, i, k.
\end{aligned}$$

Here $\mathcal{N}_t$ is the set of nodes at stage t, each with probability P_t^n, observation o_t^n, and decision a_t^n; non-anticipativity is enforced by equating decisions a_t across all scenarios sharing the same parent node.

4.6 Heuristic-Augmented Model Predictive Control

Standard MPC algorithms optimize control actions over a finite horizon N, leading to strategies that maximize short-term rewards, since the algorithm is able to "see" only N steps ahead. To overcome this limitation, we augment the MPC framework with a terminal heuristic that anticipates the system's behavior beyond the finite horizon. This extension allows the controller to account for long-term reward by incorporating an estimate of the optimal terminal observation (regarding the deterministic part) into the optimization.

We propose a method of creating the heuristic that is inspired from pattern recognition. In our approach, MPC is performed over a finite horizon of N time steps. We then extend the forecast by generating predictions for an extra interval of L time steps, conditioned on the generated observations of the horizon. These extended forecasts are used to train a predictive model (we implemented using an LSTM) that estimates the optimal terminal observation, denoted by

$$o^{\text{opt}}_{N+k_0-1} = \text{LSTM}_\theta\Big(\hat{o}^s_{k_0:N+k_0-1},\, \hat{o}^s_{N+k_0:N+k_0+L-1}\Big),$$

where θ are the parameters of the LSTM network. This optimal terminal observation guides the optimization process, taking the extended horizon into consideration, which helps the algorithm approach the long-term objective. Regarding the training of the predictive model, we apply the MPC algorithm on windows of $N + L$ time steps of observations, and retrieve the deterministic part of observation at step N. This is the observation at step N that the system must pass through, in order to achieve the optimal action sequence for a longer horizon, which is a better approximation for the infinite-horizon goal. Hence, the model is trained to predict the expected optimal observation at the end of the horizon. We propose this concept in order to handle situations where decisions must be made fast or computationally inexpensive, meaning that optimizing directly on a larger horizon is challenging.

The Heuristic Augmented MPC optimization problem is then formulated as follows:

$$\begin{aligned}
\underset{a_{k_0},\ldots,a_{N+k_0-1}}{\text{maximize}} \quad & \sum_{k=k_0}^{N+k_0-1} R\big(o^s_k, a_k\big) - \gamma \left\| \hat{o}^s_{N+k_0-1} - o^{\text{opt}}_{N+k_0-1} \right\|^2, \\
\text{subject to} \quad & o_0, o_1, \ldots, o_{k_0-1} \text{ are observed}, \\
& \hat{o}^s_{k+1} = \mathcal{F}(o^s_0, \ldots, o^s_k), \quad k = k_0, \ldots, N + k_0 - 1, \\
& a_k \in \mathcal{A}, \quad k = k_0, \ldots, N + k_0 - 1.
\end{aligned}$$

Here, $\mathcal{F}$ denotes the forecast operator as in the previous frameworks), and γ is a weighting factor that determines the significance of ending in the predicted optimal terminal state. We propose two variants of this heuristic-augmented MPC: one with a hard terminal state constraint (achieved by setting $\gamma = \infty$), where the optimization process must definitely end up in the predicted state and one with a soft constraint ($\gamma < \infty$), where the MPC algorithm can balance between achieving the terminal state and optimizing the cumulative reward. One may choose between infinite and finite γ, depending on how accurate their predictive model is.

By integrating the terminal heuristic into the MPC formulation, we indirectly extend the planning horizon beyond N time steps, so that the optimal control sequence not only maximizes short-term rewards but also positions the system favorably for future operations, mitigating the limitations of the finite-horizon optimization.

4.7 Model-Free Reinforcement Learning

To provide a holistic evaluation of learning strategies, we define various model-free RL baselines. In this way, we explore a broader set of methods on the same task. Moreover, comparing these baselines against our proposed model-based MPC framework (*D-I MPC*) will provide insight considering the value of learning an explicit probabilistic model of uncertainty versus relying purely on trial-and-error. We define three versions of model-free agents, all operating in the POMDP setting, but differing in how they represent the environment state.

4.7.1 Model-Free RL with Idealized Hidden State

To establish an upper-bound benchmark, we consider a model-free agent that (unrealistically) has access to the future K values of the stochastic process $\{o^s_{t+1}, \dots, o^s_{t+K}\}$ governing the system's uncertainty. We consider as state these future values along with the current observation:

$$s_t \;=\; \Big[o_t,\; o^s_{t+1:t+K}\Big],$$

The agent then learns a parameterized policy $\pi_\phi(a \mid s)$ directly on these "true" states, aiming to maximize the usual discounted return. By training a standard off-policy RL algorithm (e.g., a Deep Q-Network), we obtain a reference level of performance that most likely no model-free method with only partial observations can surpass.

4.7.2 Model-Free RL with LSTM Hidden State

Moving on to frameworks that can be implemented in a real scenario, when only partial observations o_t are available, we stack recent observations into a sequence $o_{t-H+1:t} = (o_{t-H+1}, \dots, o_t)$ and learn an internal hidden representation h_t via a recurrent network. Specifically, we employ a LSTM network parameterized by ψ to calculate a hidden state:

$$h_t \;=\; \mathrm{LSTM}_\psi\big(o_{t-H+1:t},\, h_{t-1}\big),$$

so that the agent's input becomes

$$s_t \;=\; \big[o_t,\; h_t\big].$$

This hidden vector h_t summarizes the history and serves as the agent's internal belief.

4.7.3 Model-Free RL with TimeGrad Hidden State

Since our research proposes using diffusion models as system models, in this version we replace the standard LSTM belief state with the hidden state learned by TimeGrad. At each time t, we pass the most recent stochastic observations $o^s_{t-H+1:t} = (o^s_{t-H+1}, \dots, o^s_t)$ through the TimeGrad, yielding a hidden embedding h^{TG}_t, just before producing the distribution of future values. This embedding captures the temporal dependencies and uncertainty encoded by the diffusion forecaster. We then concatenate h^{TG}_t with the deterministic part of the observation o^d_t to form the input feature vector for our Deep Q-Network (DQN):

$$h^{\mathrm{TG}}_t \;=\; \mathrm{TimeGradRNN}_\theta\big(o^s_{t-H+1:t},\, h_{t-1}\big),$$

which captures both temporal patterns and uncertainty via the learned diffusion encoding. The full agent state is then

$$s_t \;=\; \big[o_t,\; h_t^{\mathrm{TG}}\big].$$

Training proceeds exactly as in the LSTM case, but now the hidden component h_t^{TG} is informed by the diffusion model's internal representation of stochastic dynamics.

Chapter 5

Experimental Results

Contents

5.1 Application in Energy Arbitrage

To evaluate the proposed method, *Diffusion-Informed MPC*, we test how the integration of a diffusion-based time series forecaster, specifically TimeGrad [7], within various model-based optimization techniques, such as Model Predictive Control, performs in the context of energy arbitrage in the day-ahead electricity market of the New York Independent System Operator (NYISO) [55]. In this setting, an agent must control the operation of a Battery Energy Storage System (BESS) to maximize long-term profit. The agent has to make sequential decisions under the uncertainty of the future energy prices.

In many energy markets, including the market of New York, the efficient operation of energy storage systems plays a crucial role in electricity grid management. These systems provide flexibility and stability to the grid while enabling opportunities for profit through energy arbitrage. Energy arbitrage involves purchasing electricity during periods of low prices, storing it, and reselling it during periods of high prices, thereby exploiting price fluctuations for financial gain. This application requires the prediction of future energy prices and a decision-making strategy to maximize profit over the long term.

The market works with participants submitting bids for buying and selling electricity for the following day. Prices in this market are determined by the intersection of supply and demand curves and the participants' bids, therefore participants must predict the prices in order to plan their energy transactions and, since the goal is to maximize the long-term profit, accurate forecasting is essential for informed decision-making, so as to overcome the uncertain factors that influence prices.

The NYISO is responsible for managing the electricity grid in New York State, ensuring the reliable and transparent operation of the power system and overseeing the electricity markets. New York State is divided into several zones, each with distinct electricity prices to account for the local supply and demand. These zones range from Zone A (West) to Zone K (Long Island), with some regions exhibiting high price volatility, for example due to their denser population. We focus on a real-world energy market, so our study provides a theoretical approach that has a real-world application.

A Battery Energy Storage System (BESS) [56] is a technological solution that stores electricity during periods of surplus or low prices and discharges it during periods of high demand or high prices. These systems play a critical role in energy arbitrage, frequency regulation, peak shaving, and renewable energy integration. For the purpose of this study, we focus on the energy arbitrage function of BESS. The operation of a BESS is constrained by factors such as its state of charge (SoC), charging and discharging efficiency, and physical limits on the amount of energy that can be stored or delivered. An intelligent agent must account for these constraints while aiming to maximize profit through energy arbitrage over the long term.

Energy prices of the various regions exhibit strong temporal dependencies, seasonal patterns, and stochastic variations. Capturing these characteristics requires models capable of learning and predicting complex multivariate time series distributions. Diffusion models, as explained, are suitable for this task.

TimeGrad [7], which is a time series forecaster based on *Conditional Diffusion Models*, can model the distribution of multiple future energy prices by iteratively adding noise to the observed data and learning to reverse this process to generate realistic future samples. This approach captures the inherent uncertainty in price forecasts. Moreover, its ability to probabilistically predict future trajectories, forming a distribution of the future prices, provides with a probabilistic representation of possible future price trajectories.

To test the proposed methods, we use TimeGrad's probabilistic forecasts as a model for the future prices. This study addresses various topics, such as how effectively can diffusion models like TimeGrad forecast energy prices for NYISO's day-ahead market, how well our decision-making framework can optimize the operation of a BESS for energy arbitrage, how much better are diffusion models than other predictive models, both in forecasting, and for guiding optimization algorithms, and finally, if our Model-Based approach has benefits over Model-Free RL. Through this exploration, valuable insights are provided regarding the potential of combining advanced forecasting models and decision-making algorithms in real-world applications.

5.2 Dataset

The dataset utilized in this study is publicly available by the NYISO. This dataset consists of historical data on electricity prices in the day-ahead market, offering hourly and zonal Locational Based Marginal Prices (LBMP). The data spans six years and covers the regions across New York State, each characterized by electricity prices determined by local supply and demand dynamics.

For example, Figures 5.1 and 5.2 illustrate the energy price time series for various regions over 7 days in the winter of 2018 and 7 days in the summer of 2020, respectively. One may observe by these figures that there are various seasonal patterns and periodic trends present in the data: 1) the time series have a periodic component, 2) their trend is variable, 3) the season plays a crucial role in their morphology, 4) there are various anomalies, such as in the lower plot of Figure 5.2, and most importantly, 5) the time series correlate heavily on one another.

Basic statistics give an overview of the central tendency and dispersion of the data. Table 5.1 presents the mean, quantiles one and three, standard deviation, minimum, and maximum values for the energy prices across the regions.

Each region has 52,584 data points corresponding to hourly values recorded across six years. Key characteristics of the data include the mean prices, which range from \$24.85 in the NORTH region to \$49.42 in LONGIL, meaning there are regional price disparities. LONGIL and NYC regions generally have higher average prices and, conversely, regions like NORTH and GENESE exhibit lower average prices. The standard deviations indicate substantial variability in prices, with LONGIL (\$37.79) and HUD VL (\$30.73) showing higher volatility. This suggests these regions provide more opportunities for profit, due to the high variability of the prices. Meanwhile, regions like NORTH (\$23.40) show relatively lower volatility, implying more stable prices.

The minimum and maximum values demonstrate the extreme observed prices, that happen due to high demand or supply. We note that negative prices result from oversupply situations where it is optimal for generators to pay to release energy in order to remain operational.

The 1st percentile (Q1) and 3rd percentile (Q3) provide additional information about the price distributions. For instance, LONGIL's Q3 (\$58.19) is significantly farther from the median than Q1 (\$26.53) is, meaning that there is tendency toward higher prices. Generally this is the case in all regions, i.e. Q3 is farther from the Median than Q1 is.

We checked for missing values in the dataset to ensure data completeness, and there were no missing data, making the dataset a great choice to test our idea.

As for the data distribution and outliers, we use the Interquartile Range (IQR) method. Figure 5.3 displays box plots for energy prices across regions. We observe that the medians for the LBMPs lie in the range \$19 to \$31, so they are relatively similar to one another. Additionally, the length of the boxes is indicative of the variability of the prices of each region. Although some regions, like LONGIL, exhibit slightly higher variability in the prices than others, there are no significant differences. What is important to note is that most of the medians do not lie in the middle of the boxes, and more specifically, the medians are usually lower than the means of the prices. This means, as will be showcased later using histograms, that the distributions of the prices are positively skewed, for most regions. As a result, using the normal distribution to model the data may not be a good approximation.

The whiskers of the box plots (representing the range of data outside the 1st and 3rd quantiles), 1) do not spread out much, meaning that the values outside 25th and 75th percentile are relatively concentrated, and 2) are asymmetrical, indicating positive skewness which aligns with the observation made when examining the medians. Considering the outliers, they all lie to the

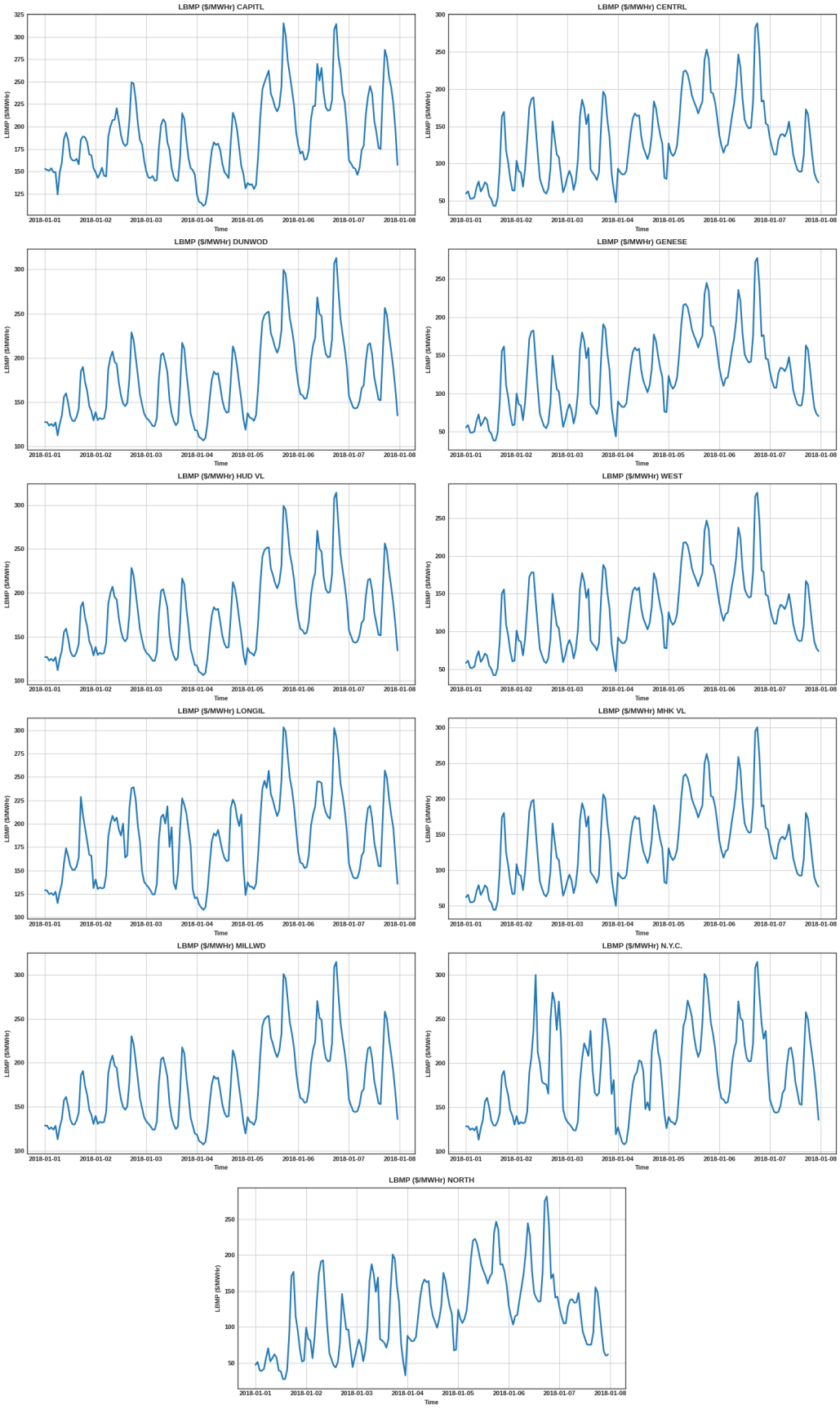


Figure 5.1: Daily fixed energy prices by NYISO for 7 days of the winter 2018.

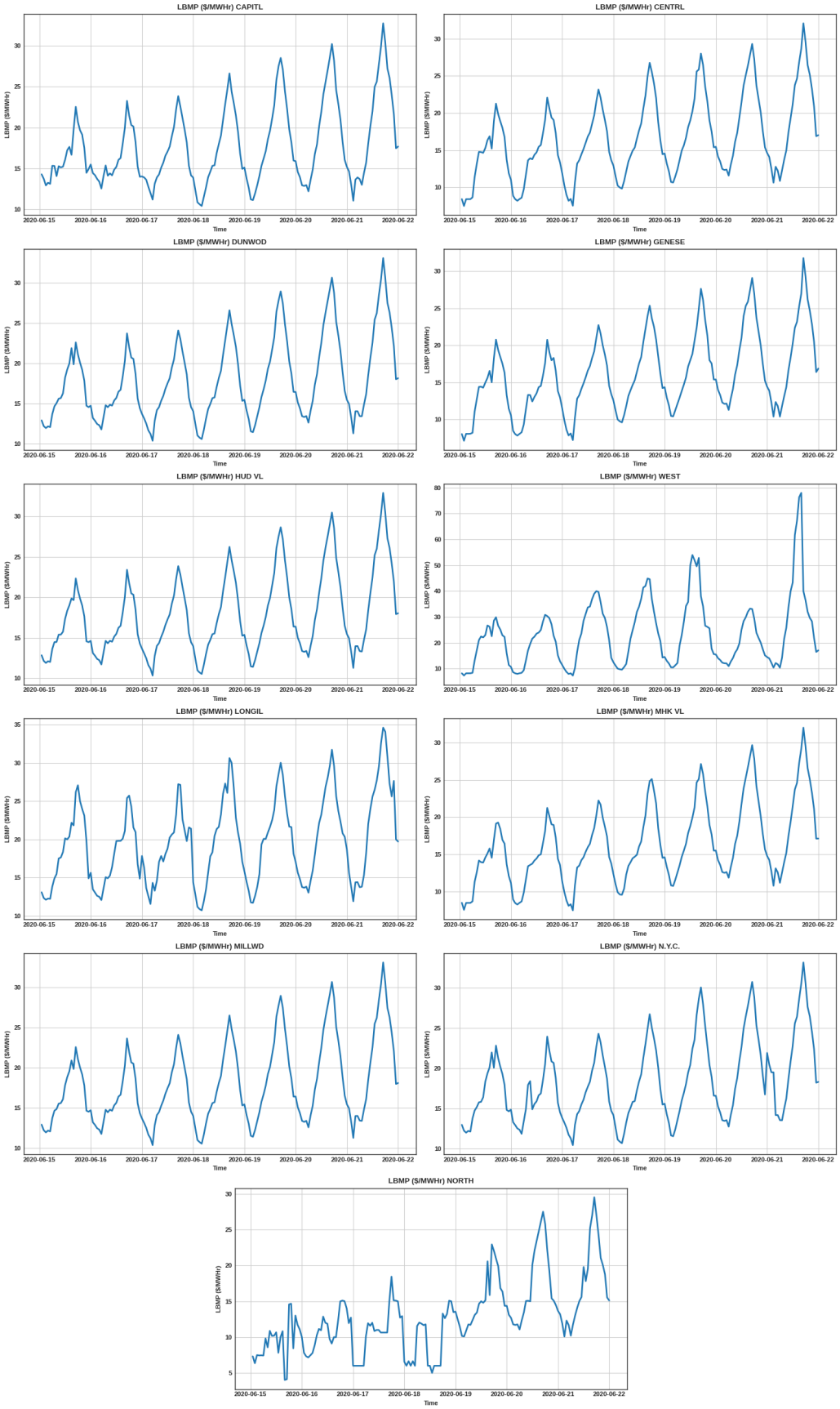


Figure 5.2: Daily fixed energy prices by NYISO for 7 days of the summer of 2020.

Region	Count	Mean	Std	Min	Q1	Median	Q3	Max
CAPITL	52,584	43.75$	37.27$	-1.41$	23.03$	31.53$	49.53$	522.77$
CENTRL	52,584	30.61$	24.33$	1.97$	16.88$	24.31$	35.58$	362.47$
DUNWOD	52,584	40.73$	31.25$	5.71$	22.33$	30.73$	47.21$	398.16$
GENESE	52,584	29.46$	23.52$	1.44$	16.06$	23.43$	34.28$	350.32$
HQ	52,584	26.39$	22.76$	-1.35$	14.10$	20.78$	31.34$	367.25$
HUD VL	52,584	40.04$	30.73$	-10.78$	22.00$	30.24$	46.44$	394.83$
LONGIL	52,584	49.42$	37.79$	9.08$	26.53$	37.26$	58.19$	569.81$
MHK VL	52,584	31.03$	25.25$	1.98$	16.89$	24.46$	36.06$	385.64$
MILLWD	52,584	40.62$	31.19$	5.68$	22.28$	30.63$	47.06$	396.70$
NYC	52,584	41.85$	32.01$	5.77$	23.09$	31.90$	48.50$	400.72$
NORTH	52,584	24.85$	23.40$	-2.91$	12.45$	19.45$	30.06$	373.13$
WEST	52,584	31.57$	25.36$	2.09$	17.17$	25.10$	36.89$	653.41$
Average		35.73$	28.45$	1.44$	20.77$	27.22$	43.59$	569.81$

Table 5.1: Basic statistics for electricity prices ($/MWh) across different regions in New York State.

right of the boxes, meaning that the "unexpected" values that occur are high, reaching $400, $500, or even $600. These extreme cases offer great opportunities for profit, hightlighting the significance of using an accurate and flexible predicting model.

Furthermore, understanding the correlation between energy prices and loads across different regions helps identify dependencies and dynamics within the market. The heatmap in Figure 5.4 visualizes the correlation matrix, where more red and darker colors indicate stronger positive correlations and more blue and darker colors represent weaker or negligible correlations.

From the heatmap, it can be observed that most regions exhibit high positive correlations with one another, price-wise and load-wise, which means that price and load fluctuations in one region are strongly dependent with those in others. This overemphasizes the need of a *multivariate* predicting model, that takes into account multiple time series to make forecasts, making *TimeGrad* an excellent choice for this application. Note that certain regions like LONGIL or N.Y.C., have unique supply-demand dynamics due to their higher urbanization, thus exhibit slightly lower correlations with more rural regions like NORTH or WEST. Also, there is no negative correlation value, meaning that all variables tend to move in the same direction.

Another way to draw some insights from the data is to decompose the time series, that is to break down the data into trend, seasonal, and residual components. For instance, the decompositions for the CAPITL region are shown in Figures 5.5 to 5.8, for the NYC region are displayed in Figures 5.9 to 5.12, and for the WEST region in Figures 5.13 to 5.16.

The time series decomposition for each region reveals patterns in prices. The observed data (top subfigures) shows the overall behavior, with NYC showing greater fluctuations than WEST, for instance.

The *Trend* component illustrates long-term price progression. *Seasonal* components capture the periodic daily, weekly, mothly, or annually patterns, with NYC showing more intense peaks during high-demand hours compared to the smoother seasonality in WEST. *Residuals*, representing noise or irregularities, are low in magnitude for all regions, except during price spikes, which occur in case of rare and high-impact events that models have a hard time predicting.

An intresting observation is that there exist two spikes in the energy price, for all regions, near the beginnings of 2022 and 2023, despite the fact that there are no spikes in the load time

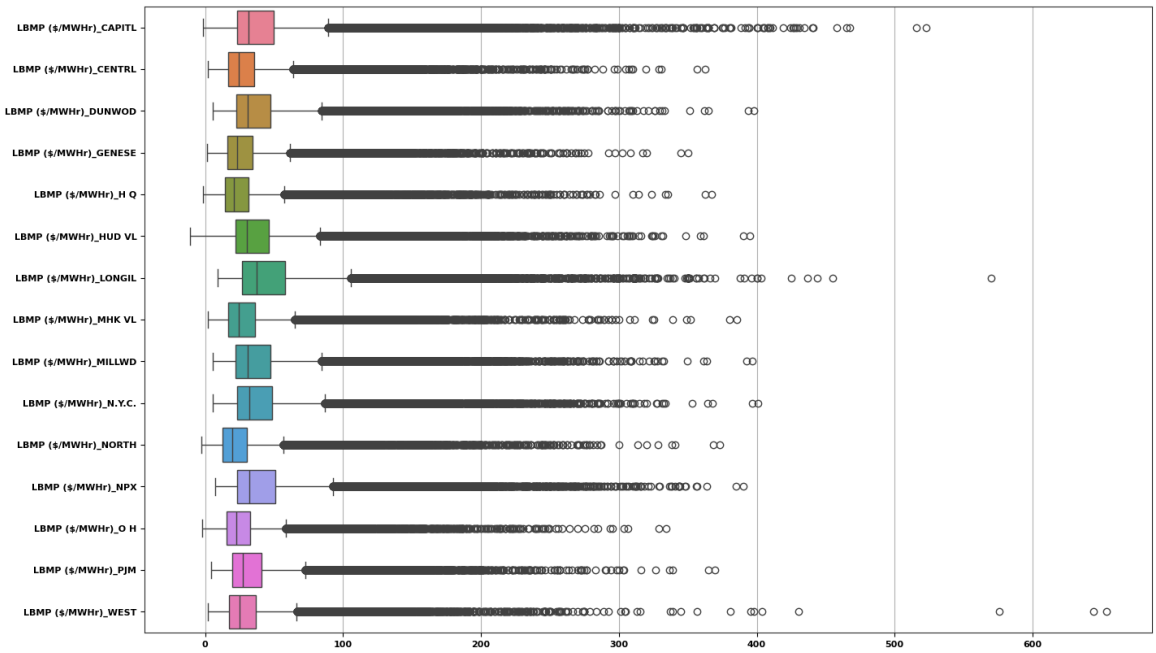


Figure 5.3: Box plots of energy prices showing outliers across various regions.

series. They can be explained through unpredictable events. According to the "EXPLAINER - Impact of National & Global Conditions on Electricity Prices in New York" [57], the spikes can be attributed to several factors, such as those outlined in the following excerpts of the report.

> *Spiking global demand for fossil fuels, lagging supply, and global instability caused by the war in Ukraine have combined to bring fossil fuel prices to historic high levels. While consumers might expect these conditions to impact the cost of gasoline, many have been surprised by the degree to which these fossil fuel prices have found their way into electricity bills as well.*
>
> *As COVID-19 restrictions eased and parts of the world began to return to normal in 2021, the demand for energy began to rise as well. This was particularly visible with the industrial and commercial sectors, which accounted for nearly 40% of natural gas consumption in the United States.*
>
> *The national inflation rate was the highest it had been in nearly 40 years. Higher fuel costs, including natural gas, led to higher average wholesale electricity prices.*
>
> *Cold weather and seasonal demand for fossil fuels to meet heating needs added to the existing demand for power generation, creating additional upward pressure on prices.*
>
> *The war in Ukraine further strained global oil and natural gas markets, adding to the economic factors.*

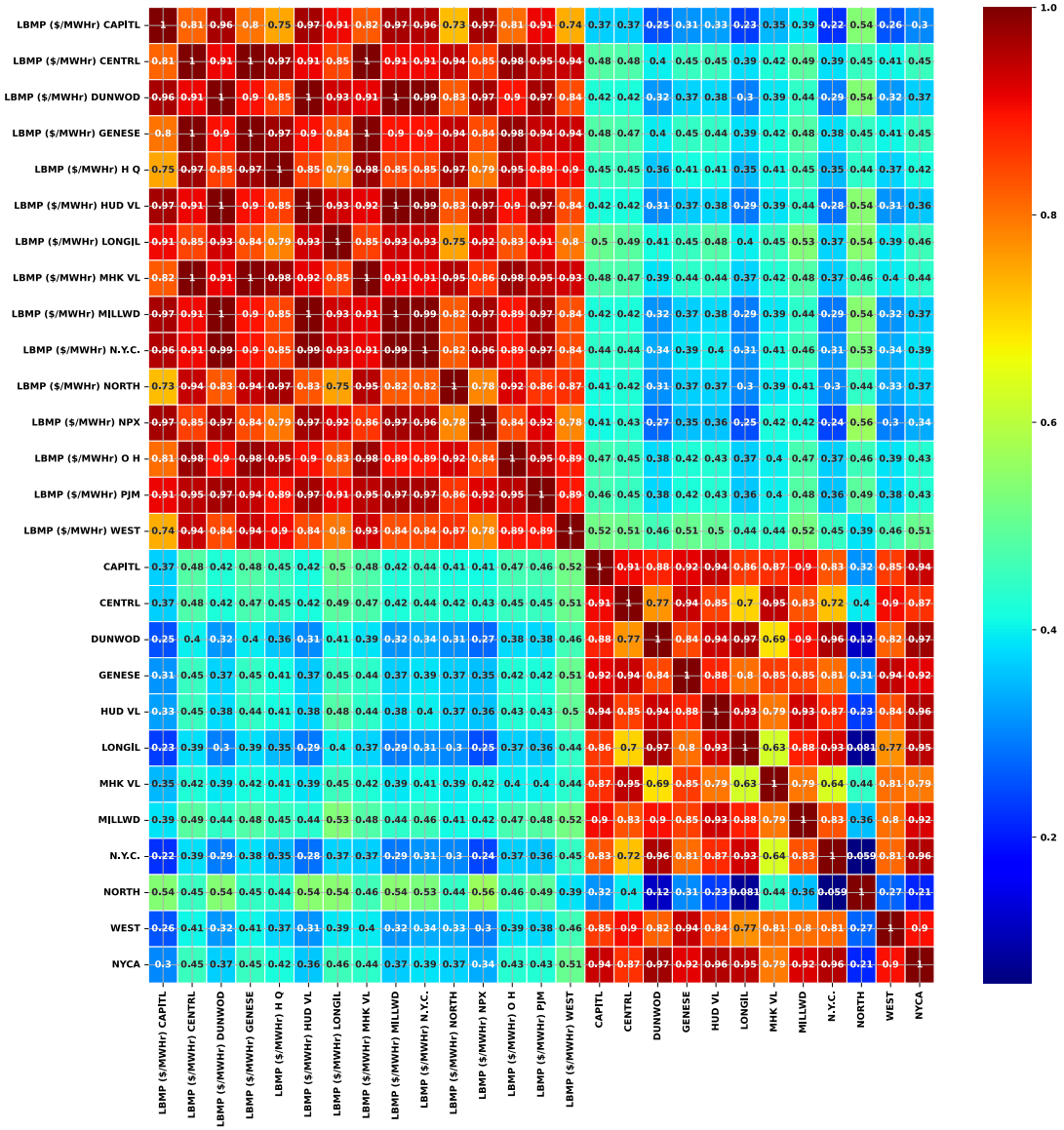
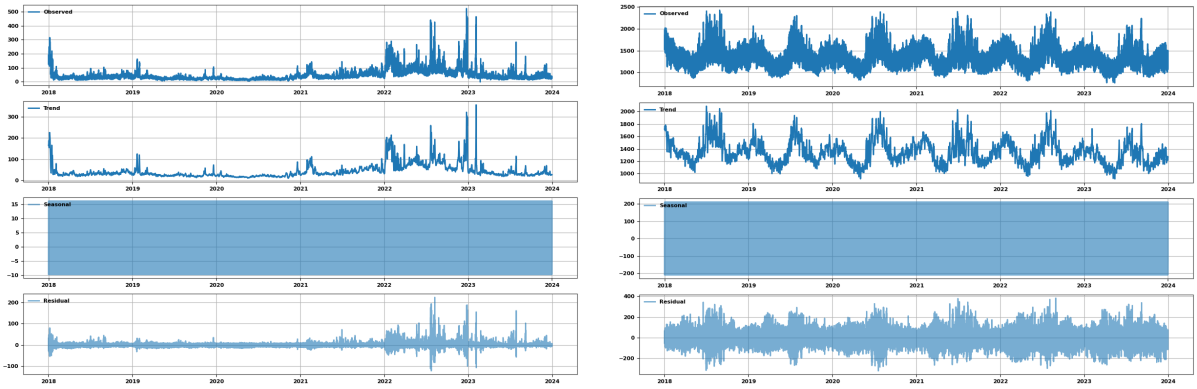
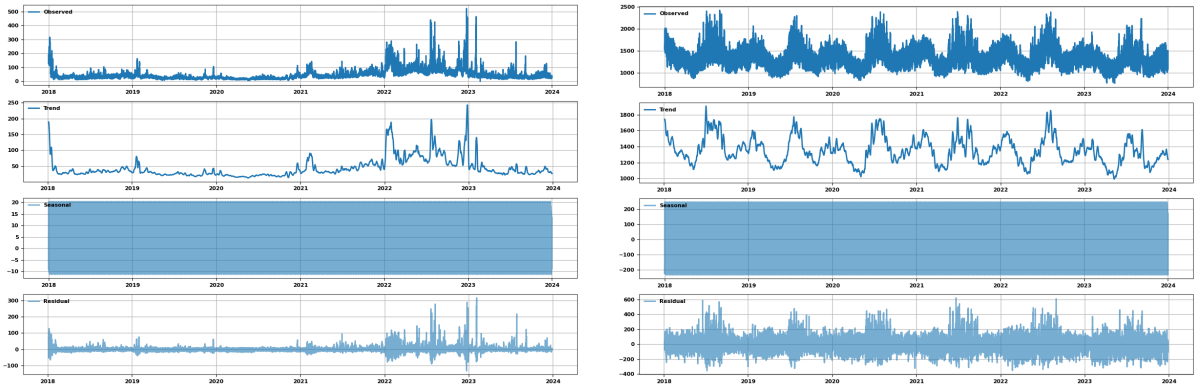


Figure 5.4: Correlation heatmap for energy prices across various regions.



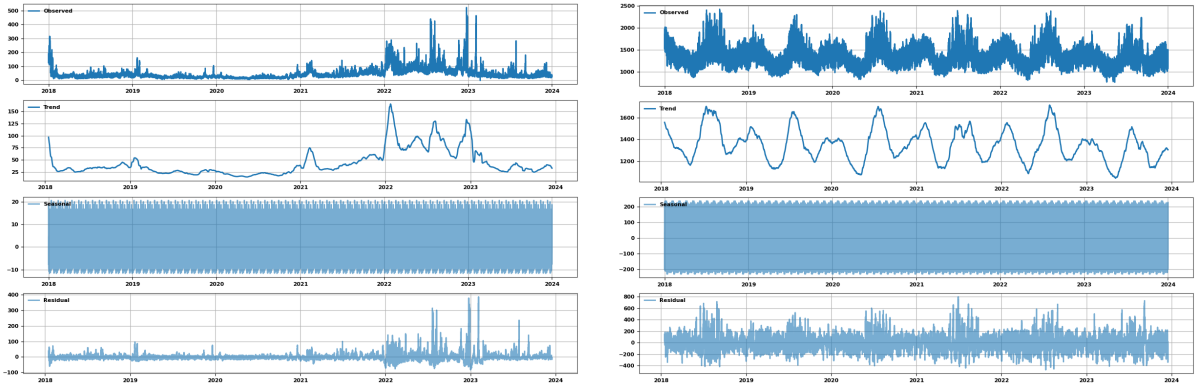
(a) Daily LBMP Decomposition for CAPITL (b) Daily Loads Decomposition for CAPITL

Figure 5.5: Daily time series decomposition for the CAPITL region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.



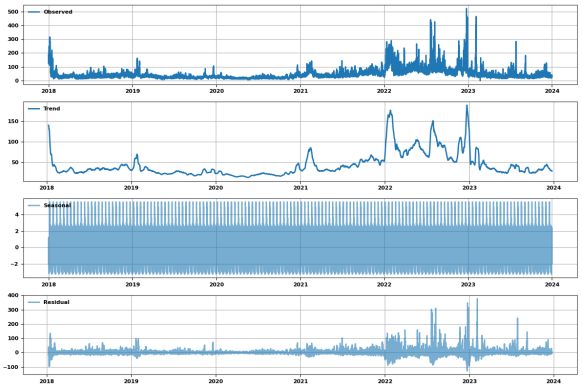
(a) Weekly LBMP Decomposition for CAPITL (b) Weekly Loads Decomposition for CAPITL

Figure 5.6: Weekly time series decomposition for the CAPITL region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.

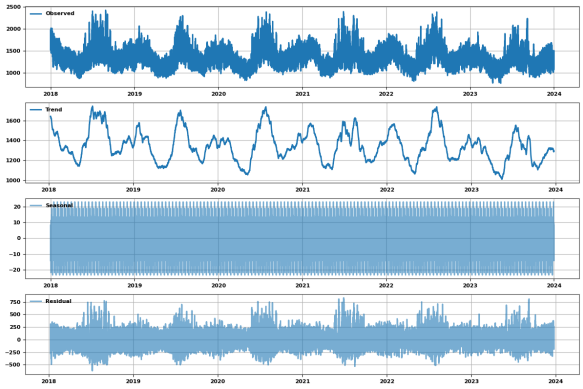


(a) Monthly LBMP Decomposition for CAPITL (b) Monthly Loads Decomposition for CAPITL

Figure 5.7: Monthly time series decomposition for the CAPITL region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.

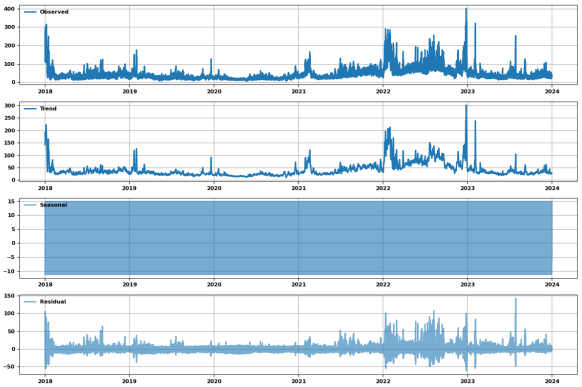


(a) Annual LBMP Decomposition for CAPITL

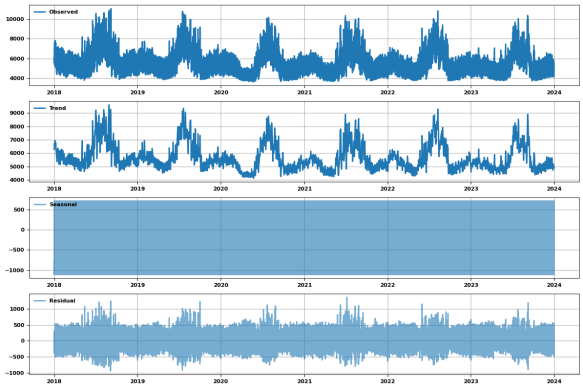


(b) Annual Loads Decomposition for CAPITL

Figure 5.8: Annual time series decomposition for the CAPITL region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.

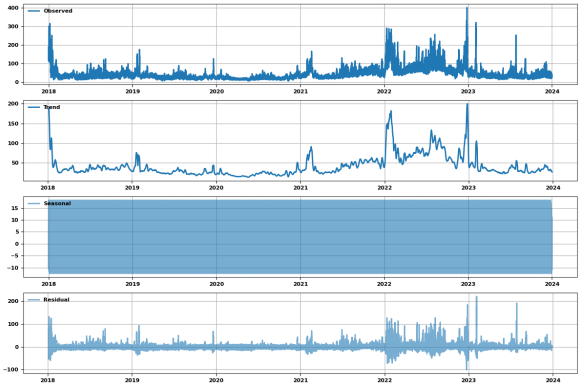


(a) Daily LBMP Decomposition for NYC

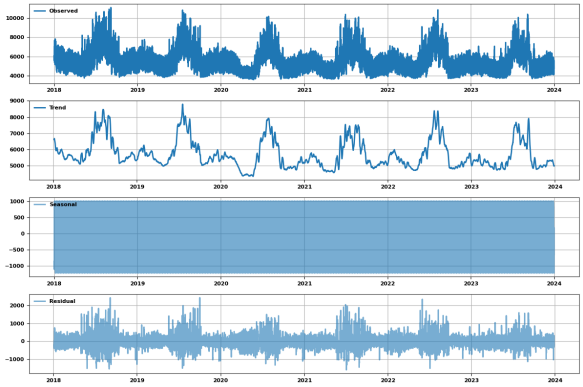


(b) Daily Loads Decomposition for NYC

Figure 5.9: Daily time series decomposition for the NYC region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.

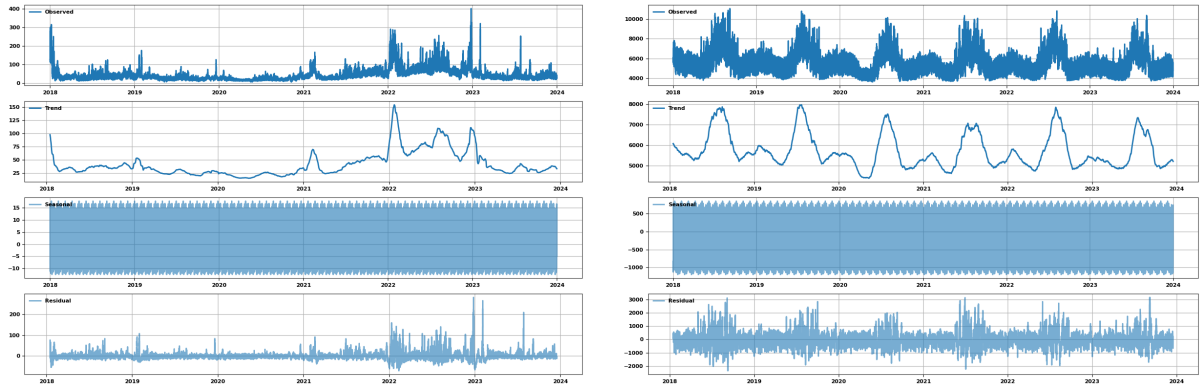


(a) Weekly LBMP Decomposition for NYC



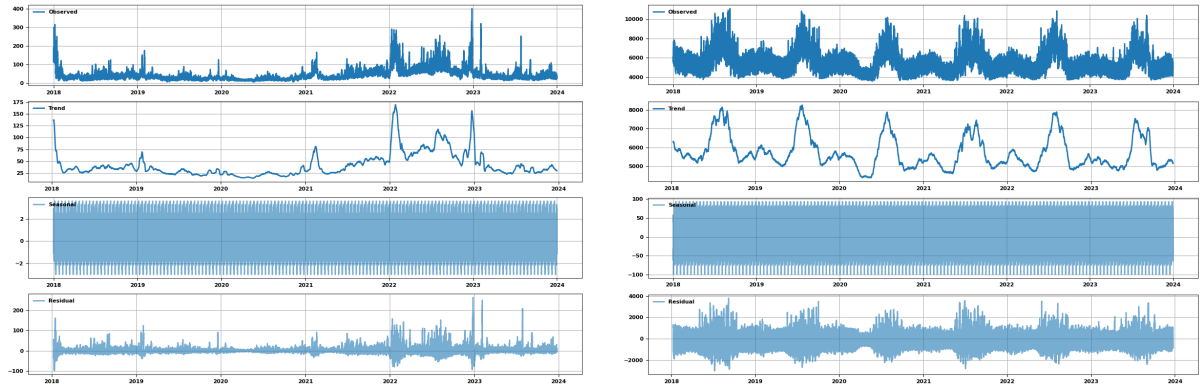
(b) Weekly Loads Decomposition for NYC

Figure 5.10: Weekly time series decomposition for the NYC region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.



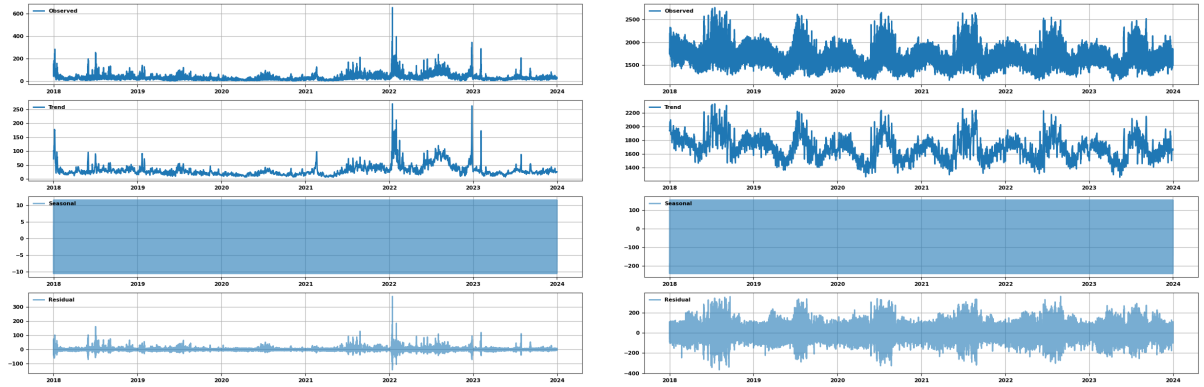
(a) Monthly LBMP Decomposition for NYC (b) Monthly Loads Decomposition for NYC

Figure 5.11: Monthly time series decomposition for the NYC region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.



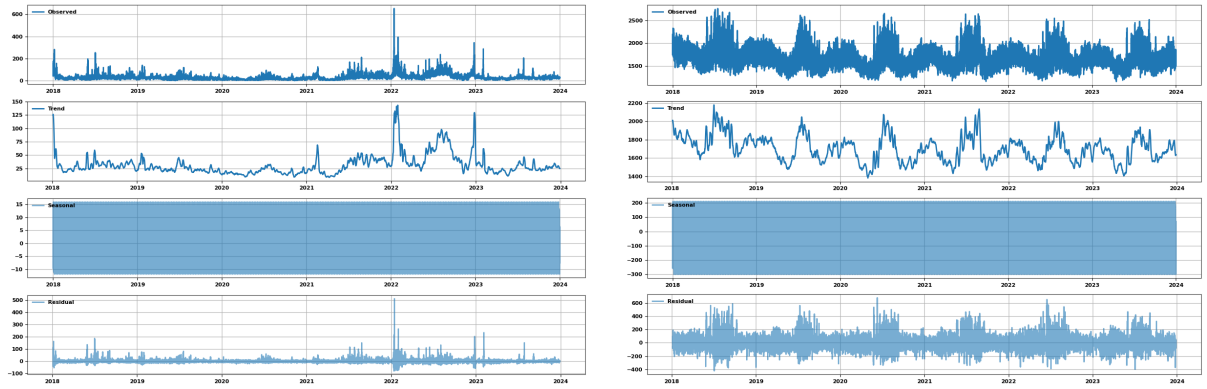
(a) Annual LBMP Decomposition for NYC (b) Annual Loads Decomposition for NYC

Figure 5.12: Annual time series decomposition for the NYC region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.



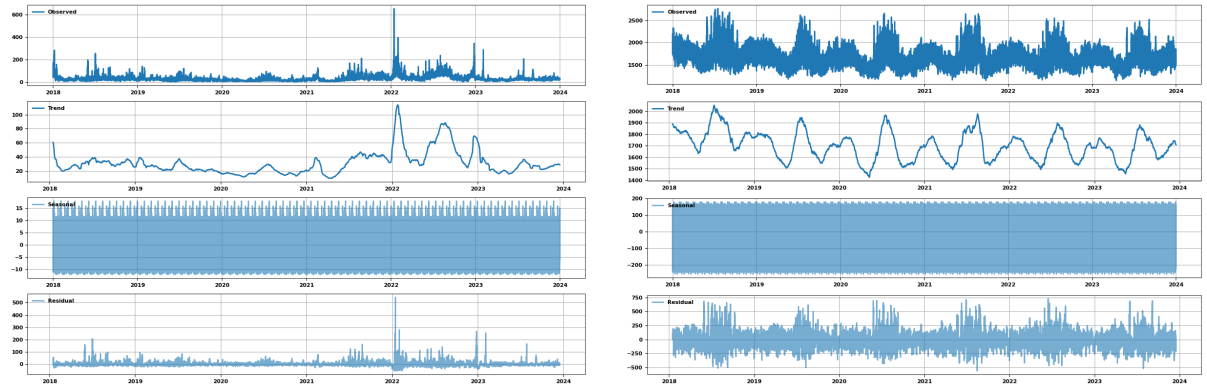
(a) Daily LBMP Decomposition for WEST (b) Daily Loads Decomposition for WEST

Figure 5.13: Daily time series decomposition for the WEST region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.



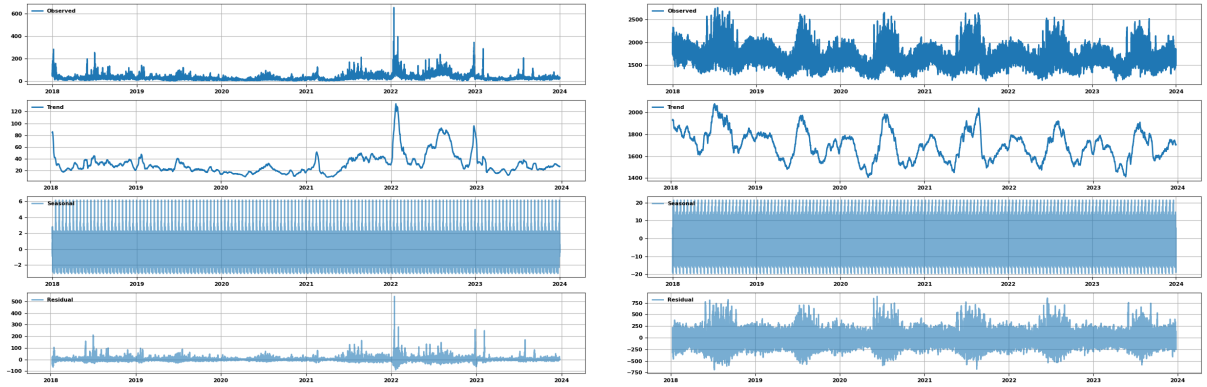
(a) Weekly LBMP Decomposition for WEST (b) Weekly Loads Decomposition for WEST

Figure 5.14: Weekly time series decomposition for the WEST region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.



(a) Monthly LBMP Decomposition for WEST (b) Monthly Loads Decomposition for WEST

Figure 5.15: Monthly time series decomposition for the WEST region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.



(a) Annual LBMP Decomposition for WEST (b) Annual Loads Decomposition for WEST

Figure 5.16: Annual time series decomposition for the WEST region. The left plot shows the decomposition of LBMP, and the right plot shows the decomposition of loads.

Autocorrelation (ACF) and partial autocorrelation (PACF) plots help in understanding the dependencies within the time series data. Figures 5.17, and 5.18 illustrate these plots for two regions. Autocorrelation measures the correlation of the time series with a lagged version of itself. Mathematically, for a time series X_t, the autocorrelation at lag k is defined as:

$$\rho_k = \frac{\sum_{t=1}^{T-k}(X_t - \bar{X})(X_{t+k} - \bar{X})}{\sum_{t=1}^{T}(X_t - \bar{X})^2},$$

where T is the total number of observations, $\bar{X}$ is the mean of the series, and k is the lag.

Regarding the *autocorrelation* plots, we can observe that they are wave-like graphs with high points every 24 time steps (e.g., 0, 24, 48) and low points at the middles of these intervals (e.g., 12, 36), indicating seasonality with a periodic cycle of 24 time steps. The significant peaks at lags 0, 24, 48, etc., indicate a strong positive correlation with data points spaced 24 time steps apart, that is a daily cycle where the values follow a similar trend every 24 hours. The low correlation values at 12, 36, etc., suggest a weaker relationship between points separated by a half cycle, one and a half cycles, etc. This occurs because the data exhibits contrasting behavior (e.g., low and high energy demand) at those intervals. Moreover, the gradual fading of the wave is explained by the fact that the influence of past cycles to current ones decreases as the lag increases. This is expected since data has additional random components and seasonal patterns lose strength over time (for example, the patterns of the cooler months do not affect the ones of the warmer months).

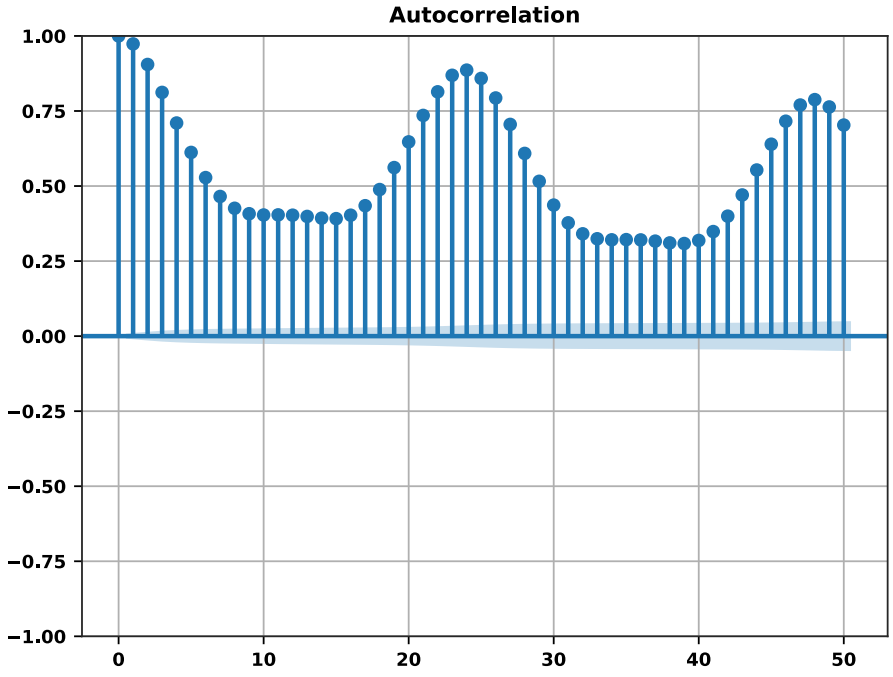
Partial autocorrelation measures the correlation between the time series and its lagged values, after removing the correlations at all shorter lags. For a time series X_t, the partial autocorrelation at lag k is the correlation between X_t and X_{t+k} after removing the effects of all lags less than k. Mathematically, for a time series X_t, the partial autocorrelation at lag k can be obtained by fitting an autoregressive (AR) model of order k and examining the coefficient of X_{t+k}. The partial autocorrelation plot in Figure 5.17 shows that the first 4 peaks are higher than the rest, suggesting strong short-term dependencies. There exist peaks around regular intervals (e.g., 24, 48) indicate seasonality with a periodic cycle of 24 time steps. A sharp cutoff followed by a gradual fading after the first 4 lags shows that there is a more complex structure, where long-term dependencies exist but weaken over time.

For Figure 5.18, the partial autocorrelation plot shows that the first 3 peaks are significant peaks are higher than the rest, meaning that dependencies exist in shorter windows than in the case of CAPITL region. We can observe stronger peaks at lags 24, 48, etc., meaning daily seasonality. A sharp cutoff followed by points with no apparent structure and of low magnitude, after the first 3 lags shows that the time series structure is even more complex than the one of CAPITL region.

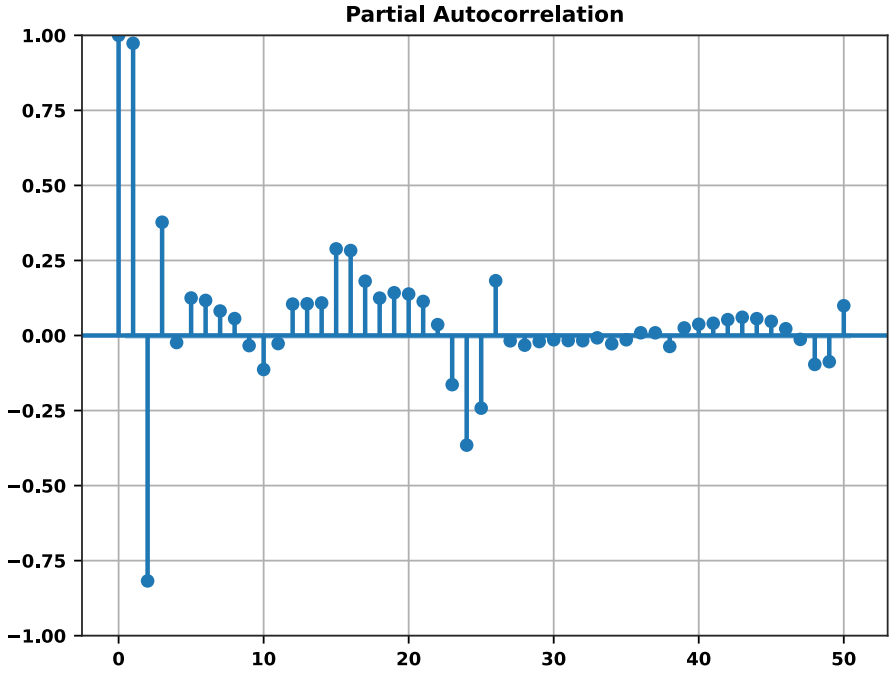
This analysis indicates that although there is strong periodicity, as expected, the underlying dependencies are complex and require a powerful model to capture them.

The Augmented Dickey-Fuller (ADF) test was performed on the time series data for electricity prices across various regions. The test evaluates the stationarity of the data and the results are provided in Table 5.2.

The ADF test evaluates the null hypothesis (H_0) that the time series has a unit root, implying it is *non-stationary*. The alternative hypothesis (H_1) suggests the series is *stationary*. The results for all regions indicate that the null hypothesis of non-stationarity is rejected, as the ADF statistic is negative for all regions and all p-values are significantly below the common threshold of 0.05. This means that we can be over 95% confident that each region's price series is truly

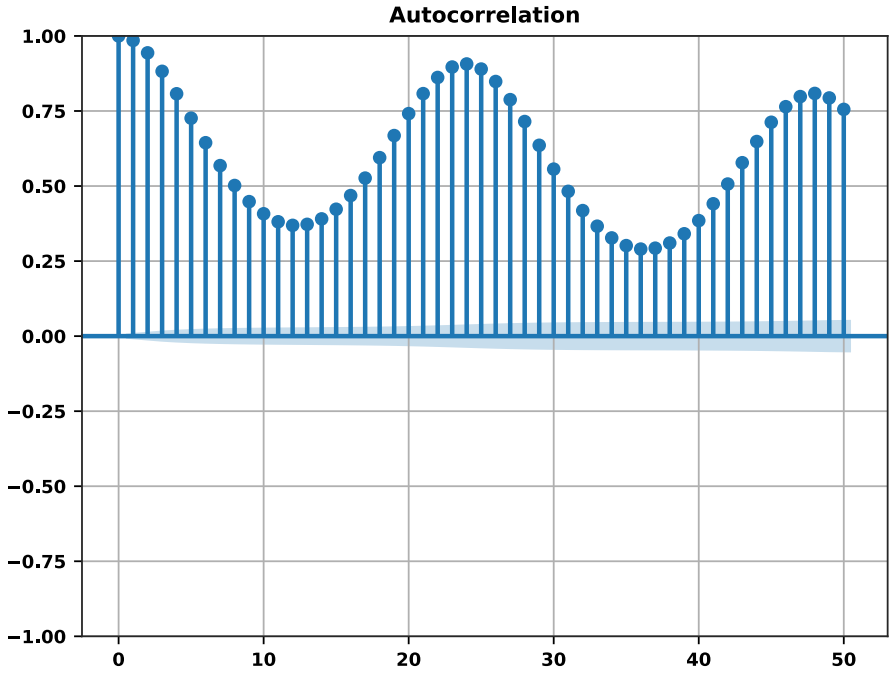


(a) ACF for CAPITL

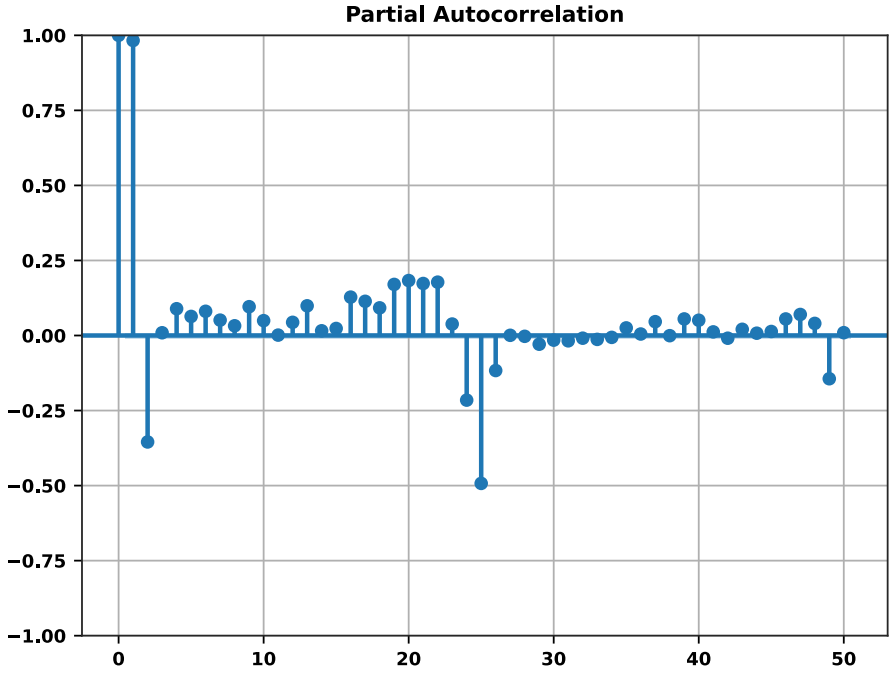


(b) PACF for CAPITL

Figure 5.17: Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots for the CAPITL region.



(a) ACF for NYC



(b) PACF for NYC

Figure 5.18: Autocorrelation (ACF) and Partial Autocorrelation (PACF) plots for the NYC region.

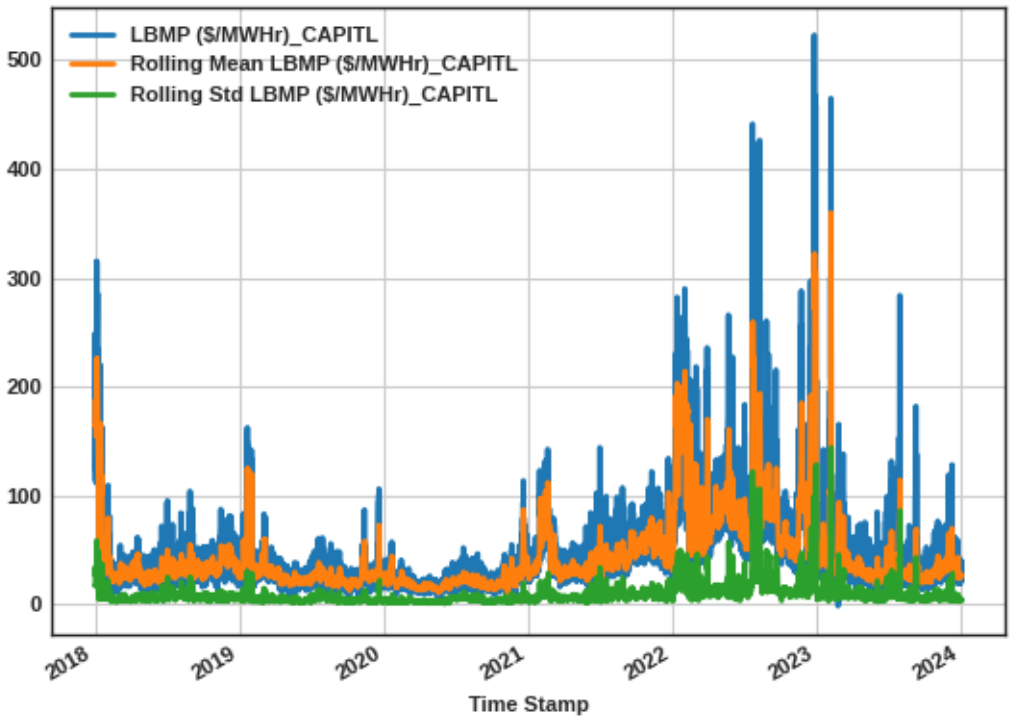
Region	ADF Statistic	p-value	Lags Used	Observations Used
CAPITL	-12.0835	2.20×10^{-22}	51	52,532
CENTRL	-13.1605	1.31×10^{-24}	57	52,526
DUNWOD	-11.1055	3.77×10^{-20}	58	52,525
GENESE	-14.8843	1.59×10^{-27}	58	52,525
HUD_VL	-11.7883	9.97×10^{-22}	54	52,529
LONGIL	-9.0261	5.60×10^{-15}	58	52,525
MHK_VL	-10.8888	1.24×10^{-19}	57	52,526
MILLWD	-12.7147	1.01×10^{-23}	58	52,525
NYC	-9.6140	1.78×10^{-16}	58	52,525
NORTH	-5.9232	2.48×10^{-7}	57	52,526
NYCA	-10.7316	2.97×10^{-19}	58	52,525
WEST	-14.7668	2.36×10^{-27}	58	52,525

Table 5.2: Augmented Dickey-Fuller (ADF) Test Results

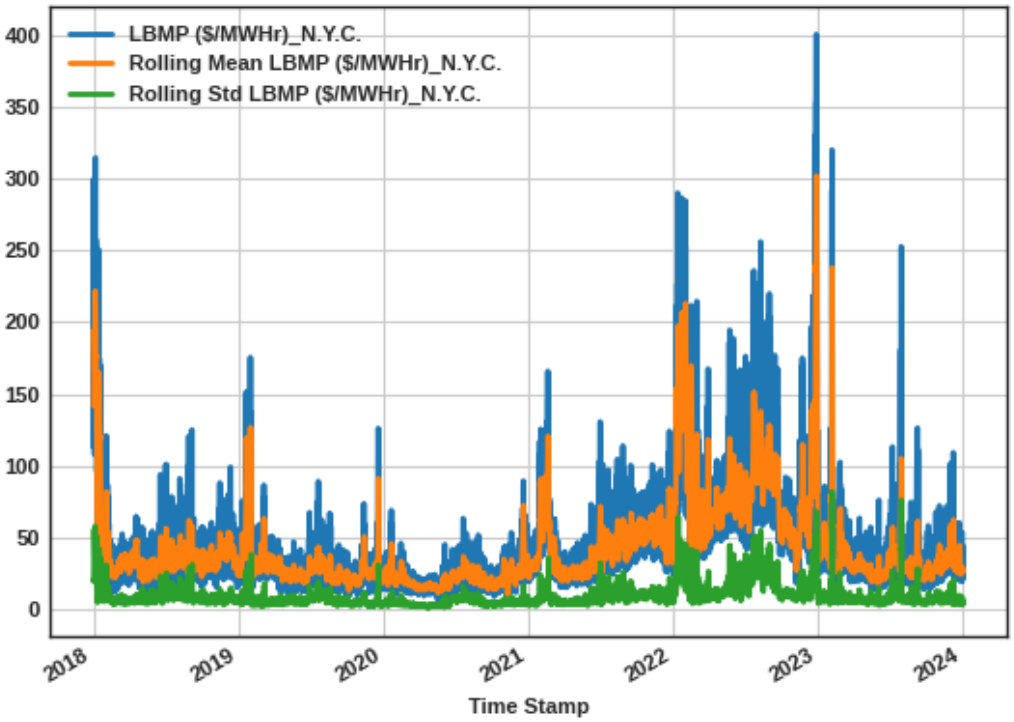
stationary. Thus, the time series data are stationary, that is their statistical properties, such as mean, variance, and autocorrelation, remain constant over small windows. The stationary nature of the time series indicates that training a model to predict them will likely succeed, however there are still intricate patterns and abnormalities in the data that require complexity in order to be understood.

Rolling statistics provide insights into how the mean and standard deviation of the data change over time. Figures 5.19a and 5.19b show the rolling mean and standard deviation for two regions. One can observe that the rolling statistics are smoothly variating, in general, reinforcing the stationarity result by the statistic. However, there are some anomalies (e.g. spikes) in the rolling mean and variance, which mean that the dynamics can be complex, hence a carefully tuned and powerful model is needed.

Histograms provide a visual representation of the distribution of data. Figure 5.20 shows the histograms for energy prices across various regions. The distributions seem positively skewed, as expected by the boxplot analysis, with some regions having higher variability. Moreover, the distributions appear to be unimodal, hence simpler for modeling (than in the multimodal case).



(a) Rolling Statistics for CAPITL



(b) Rolling Statistics for NYC

Figure 5.19: Rolling mean and standard deviation for CAPITL and NYC regions.

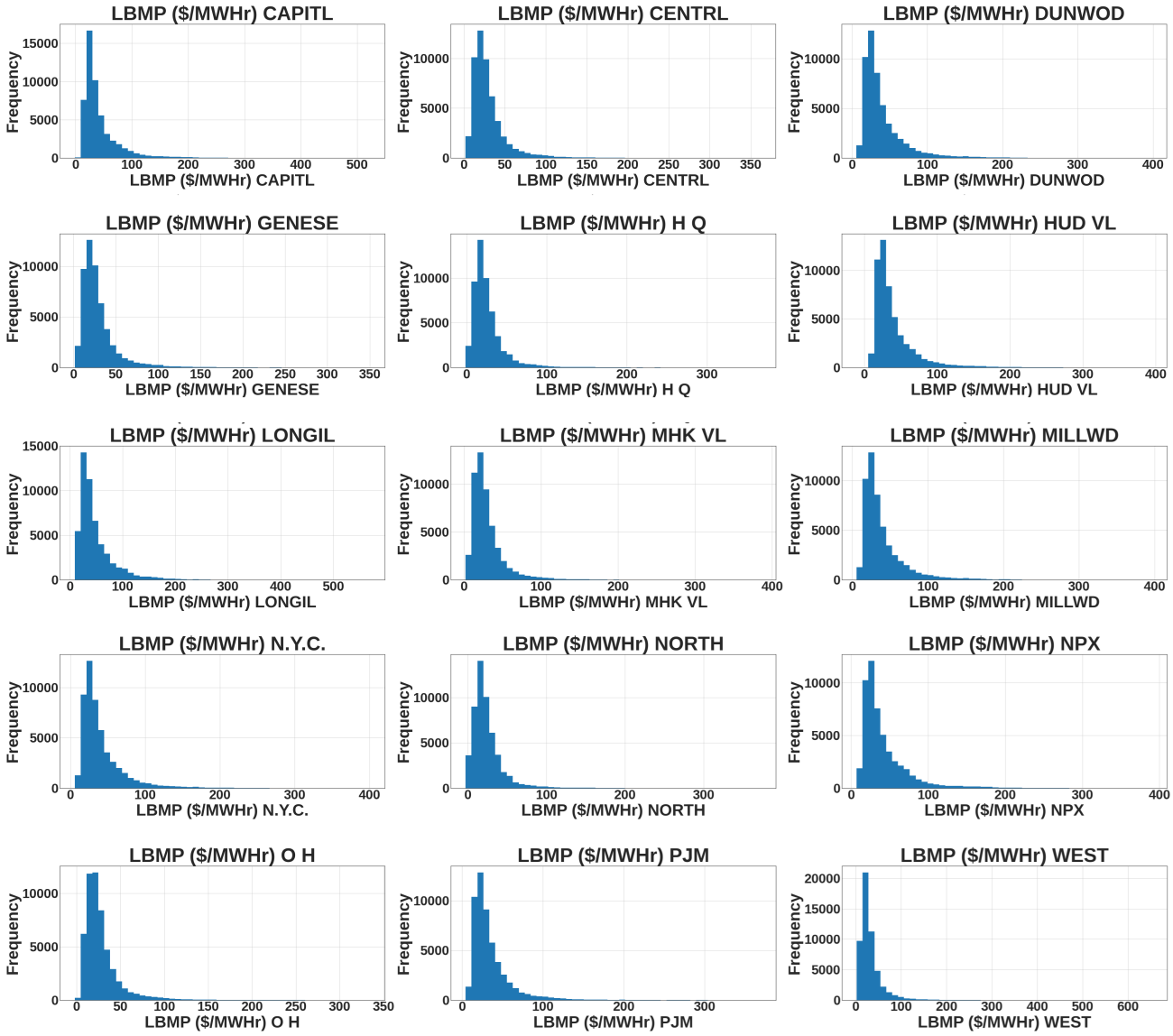


Figure 5.20: Histograms of energy prices across the regions.

5.3 Forecasting

We use TimeGrad to forecast energy prices including both LBMPs and regional load data, given their interdependencies. TimeGrad is configured to handle 27 time series, denoted as $\mathbf{x}$, modeling the probability distribution of their future values over various horizons. Mathematically, TimeGrad models the conditional distribution of future time steps based on past data, which is:

$$q(\mathbf{x}_{t_0:T}|\mathbf{x}_{1:t_0-1}, \mathbf{c}_{1:T}) = q(\mathbf{x}_{t_0:T}|\mathbf{x}_{1:t_0-1}) = \prod_{t=t_0}^{T} q(\mathbf{x}_t|\mathbf{x}_{1:t-1}),$$

where $\mathbf{x}_t$ is the multivariate vector of the time series at time t, and $c_{1:T}$ are covariates, that can encode additional information about events, such as extreme weather conditions or the news, however in our application we will not use them, as the purpose is to experiment with the forecaster and test our method of integrating it with a control framework. The model uses an RNN-based architecture to encode temporal dependencies, where the hidden state at each time step is influenced by the previous time step's data:

$$\mathbf{h}_t = \text{RNN}_\theta(\text{concat}(\mathbf{x}_t, \mathbf{c}_t), \mathbf{h}_{t-1}) = \text{RNN}_\theta(\mathbf{x}_t, \mathbf{h}_{t-1}),$$

where $\mathbf{h}_t$ is the hidden state at time t. Future steps are sampled autoregressively, with the parameters of the RNN and the diffusion model included in the process:

$$p_\theta(\mathbf{x}_t|\mathbf{h}_{t-1}).$$

The goal of TimeGrad is to approximate the conditional distribution of future time steps based on past data:

$$q(\mathbf{x}_{t_0:T}|\mathbf{x}_{1:t_0-1}) = \prod_{t=t_0}^{T} q(\mathbf{x}_t|\mathbf{x}_{1:t-1}) \approx \prod_{t=t_0}^{T} p_\theta(\mathbf{x}_t|\mathbf{h}_{t-1}).$$

Here, θ includes the parameters of the RNN and the diffusion model.

After careful tuning, we ended up with the following setup. The input size consists of 27 variables, each representing a distinct time series. For our application, the prediction length needs to be a multiple of 24 hours (e.g., 24, 48) to align with day-ahead market operations. The context length, which is the overall span of historical data considered for making predictions, was chosen to be seven days of past data. This duration was found experimentally to provide an adequate model for capturing dependencies without being overly complex in terms of training, inference, and memory requirements. The lag sequence, referring to specific past time steps used as input features for predictions, consists of two days of lagged data. We chose the DEIS (Diffusion Exponential Integrator Sampler) Multistep Scheduler to guide the diffusion process with 150 timesteps. It is a high-order solver for ordinary differential equations (ODEs) and is designed to speed up the sampling process while maintaining high sample quality. The RNN component, which encodes temporal dependencies, is configured with a LSTM architecture featuring two layers, 128 hidden units, and a dropout rate of 0.1. We trained the model using 5×10^{-5} as learning rate, for 200 epochs, and we configured the model to generate 1000 samples of predictions. This high number of samples ensures dense predicted distributions, which are necessary for certain applications and will be discussed in later sections. The embedding dimension is 5, and determines the size of the vector space in which features are embedded, the conditioning length 100 timesteps, and regarding the residual network, it has 8 layers with 8

channels and a dilation cycle length of 2. Lastly, the loss function for the training is the mean squared error with a linear beta schedule for noise variance.

TimeGrad was applied to forecast NYISO energy prices, trained on six month windows of data and tested in three month windows, so the model's performance is tested across various seasons in the period 2018–2022. The model showcased incredible prediction accuracy for future hourly prices for up to 10 days of hourly data (240 time steps in the future), generating a probabilistic distribution of trajectories. Figures 5.21 to 5.30 illustrate TimeGrad's ability to accurately forecast up to 240 time steps (10 days) across 27 time series. However, there are anomalies in the time series -caused by unexpected events or data imperfections- which TimeGrad cannot overcome in this configuration. The use of covariates could help mitigate these issues, so does the incorporation of additional data, such as the weather and the news. These problems belong to other interesting research areas, like anomaly detection.

TimeGrad was evaluated against various classical and deep learning models, including AR, ARIMA, SARIMA, ETS, CNN, and LSTM. It outperformed all models across all metrics (Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Continuous Ranked Probability Score (CRPS)). TimeGrad demonstrates impressive accuracy, even in large prediction horizon setups. This is because it is not a point-estimating autoregressive model; it learns the probability distribution of the next time step, given the distribution of the previous ones, for N days of hourly future time steps. Therefore is much more resilient to noise than a point-estimation model, where one bad point forecast would affect severely the future autoregressive forecasts. Figure 5.31 compares TimeGrad's forecasts for various horizons, focused on one region of the grid. This approach allows it to predict accurately even $10 \cdot 24 = 240$ points in the future for 27 time series simultaneously. However, there are always some anomalies due to unexpected events or dataset imperfections (see Figures 5.32 and 5.33) that are impossible to predict by any model.

TimeGrad was compared with several classical models such as AR (AutoRegressive), ARIMA (AutoRegressive Integrated Moving Average), SARIMA (Seasonal ARIMA), ETS (Exponential Smoothing State Space Model), VAR (Vector AutoRegressive), and deep learning models like CNN (Convolutional Neural Network) and LSTM. AR predicts the value of a time series based on previous values but is limited to handling linear dependencies. ARIMA extends AR by adding integration to handle non-stationarity and moving average components to capture past forecast errors. SARIMA incorporates seasonal components, making it suitable for time series with periodic trends. ETS captures trends and seasonality using exponential smoothing. VAR extends the AR model to work with multivariate time series by predicting each variable using past lags of all variables, capturing interdependencies among multiple time series. CNN models help capture local patterns in time series, detecting short-term temporal dependencies but struggling with long-term forecasting. TimeGrad outperformed these methods, showcasing its ability to model complex patterns over long horizons.

The evaluation metrics provide insights into models' performances. MSE, emphasizes larger errors and is sensitive to outliers. RMSE, as the square root of MSE, provides results in the same units as the original data. MAE, the average magnitude of errors, describes deviation, weighing the errors in the same way. MAPE expresses MAE as a percentage, for better interpretation across different scales, though it is sensitive to small actual values. CRPS evaluates how well the predicted distribution fits the actual distribution of the stochastic process (CRPS is thoroughly explained in 3.2.4).

The evaluation is summarized in Table 5.3, where TimeGrad consistently achieves the best performance across all metrics. Trained on six months of hourly data and tested on one month

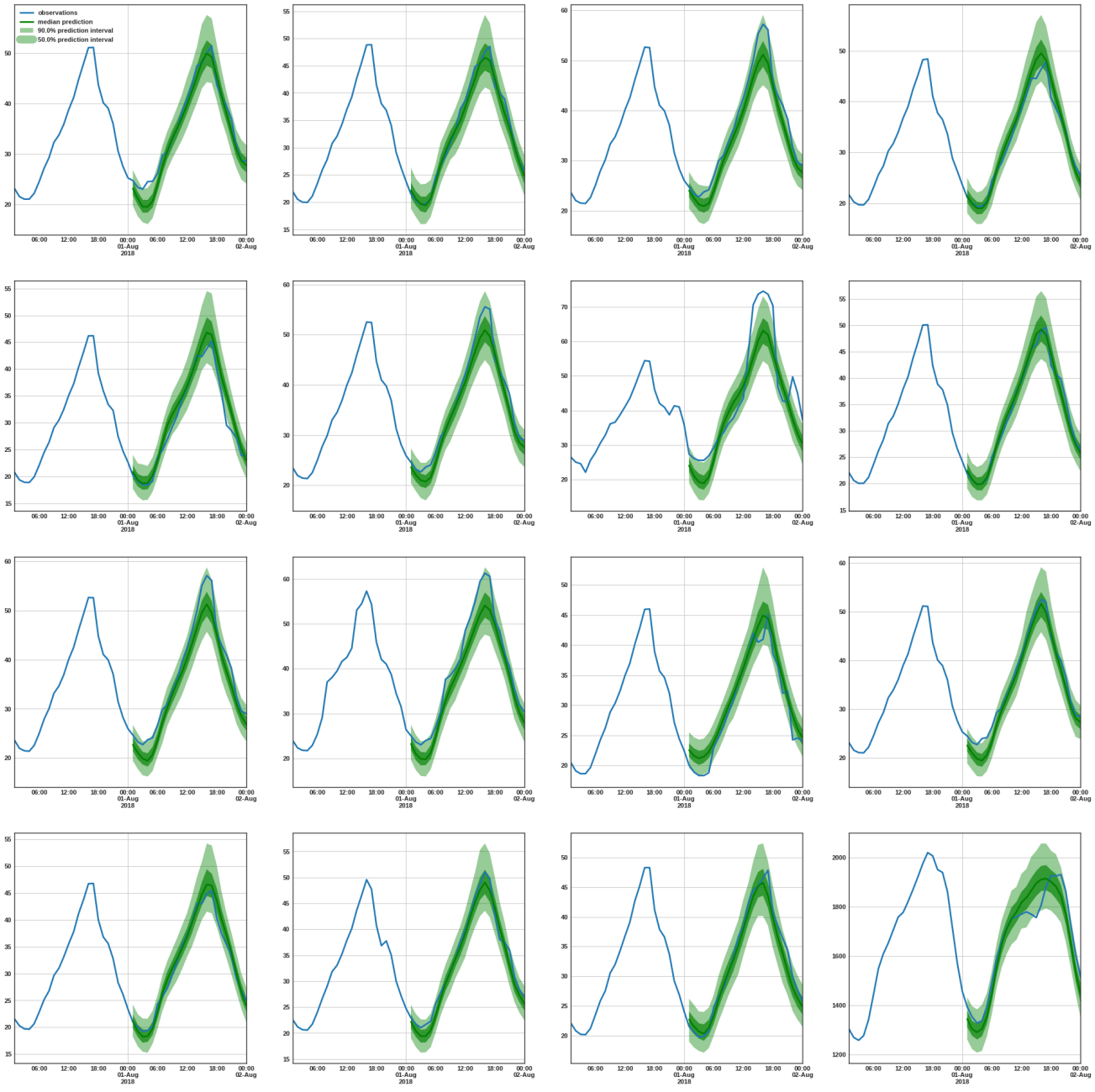


Figure 5.21: TimeGrad's hourly probabilistic forecasts for 1-day horizon.

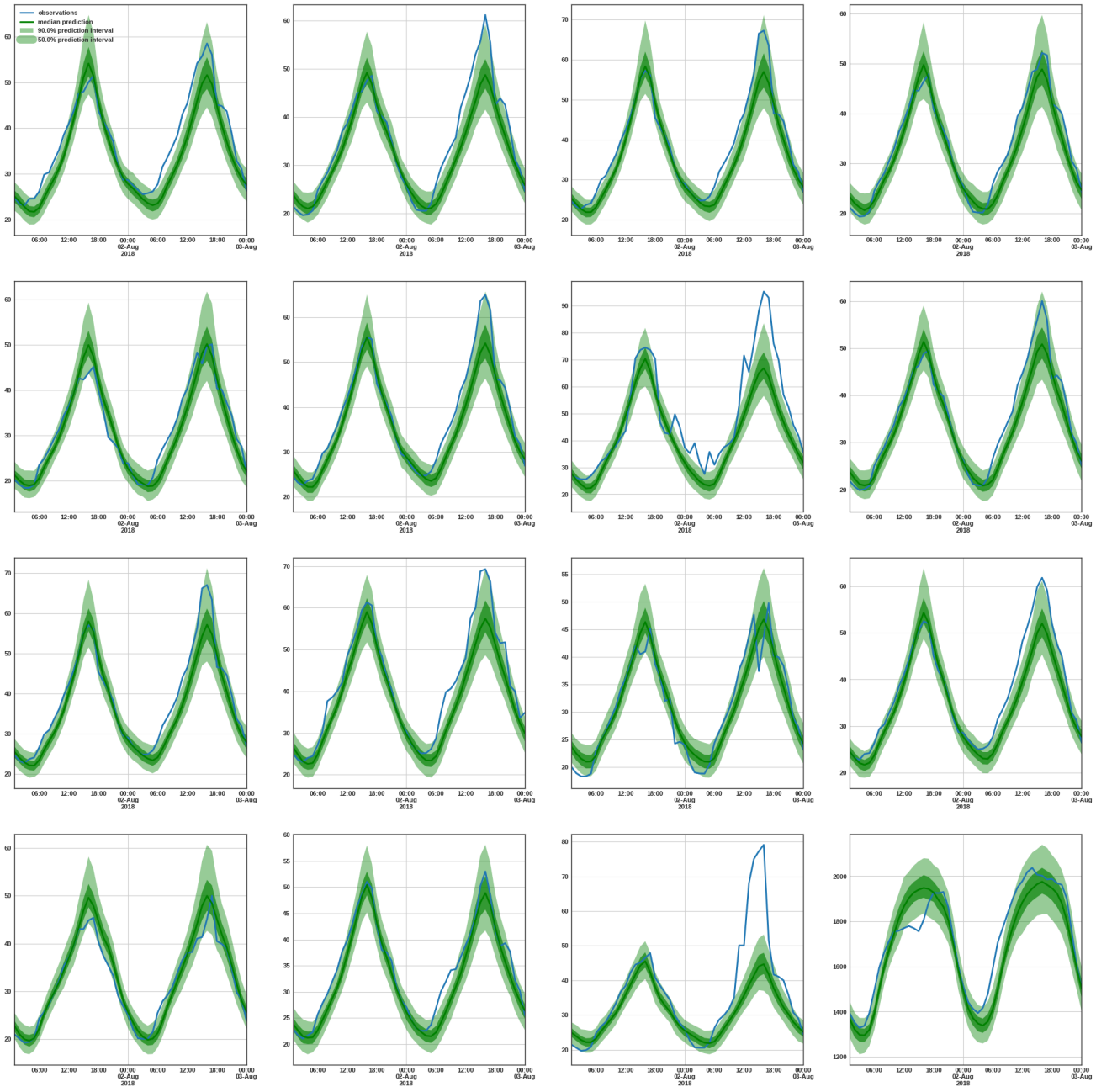


Figure 5.22: TimeGrad's hourly probabilistic forecasts for 2-days horizon.

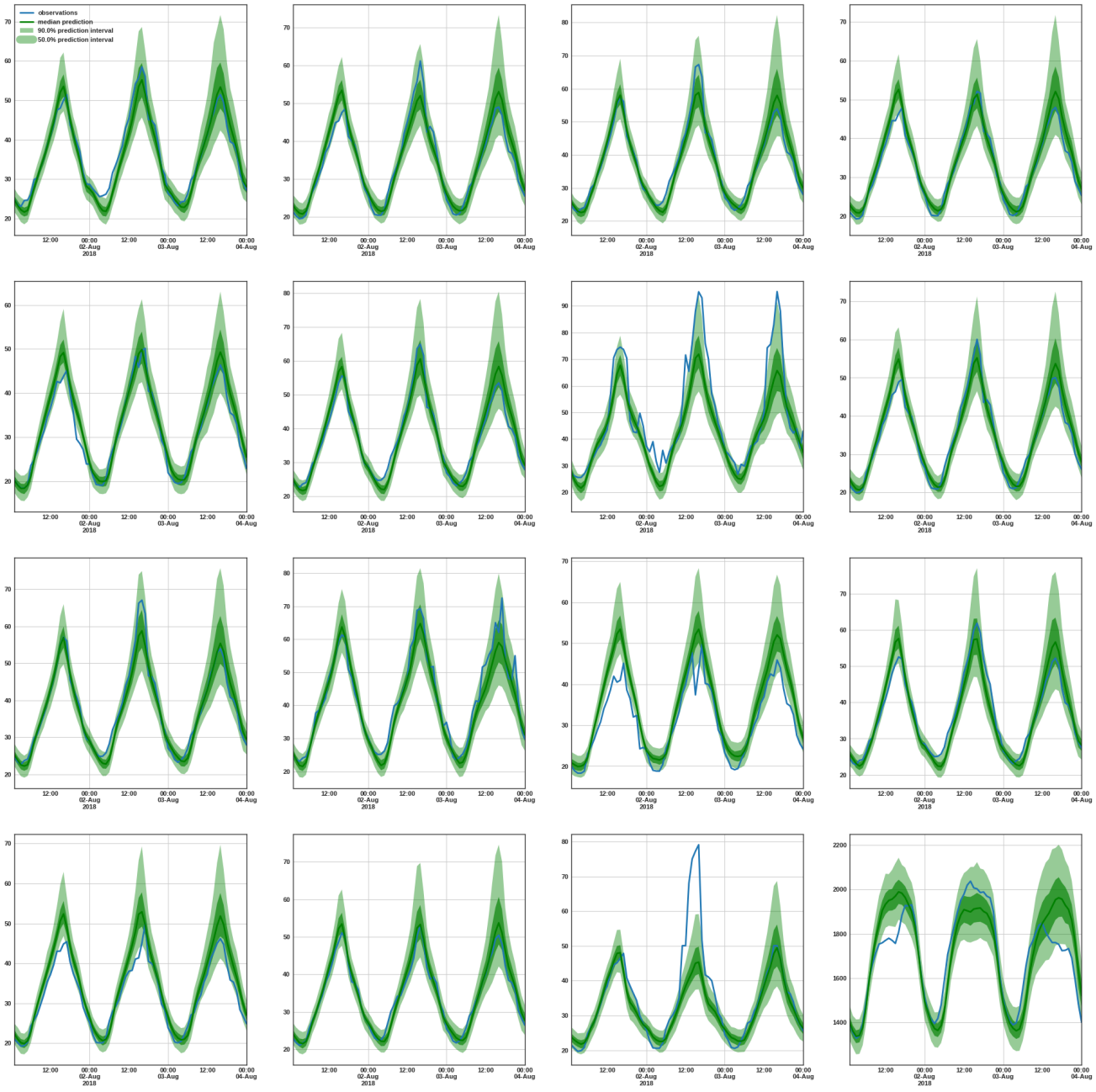


Figure 5.23: TimeGrad's hourly probabilistic forecasts for 3-days horizon.

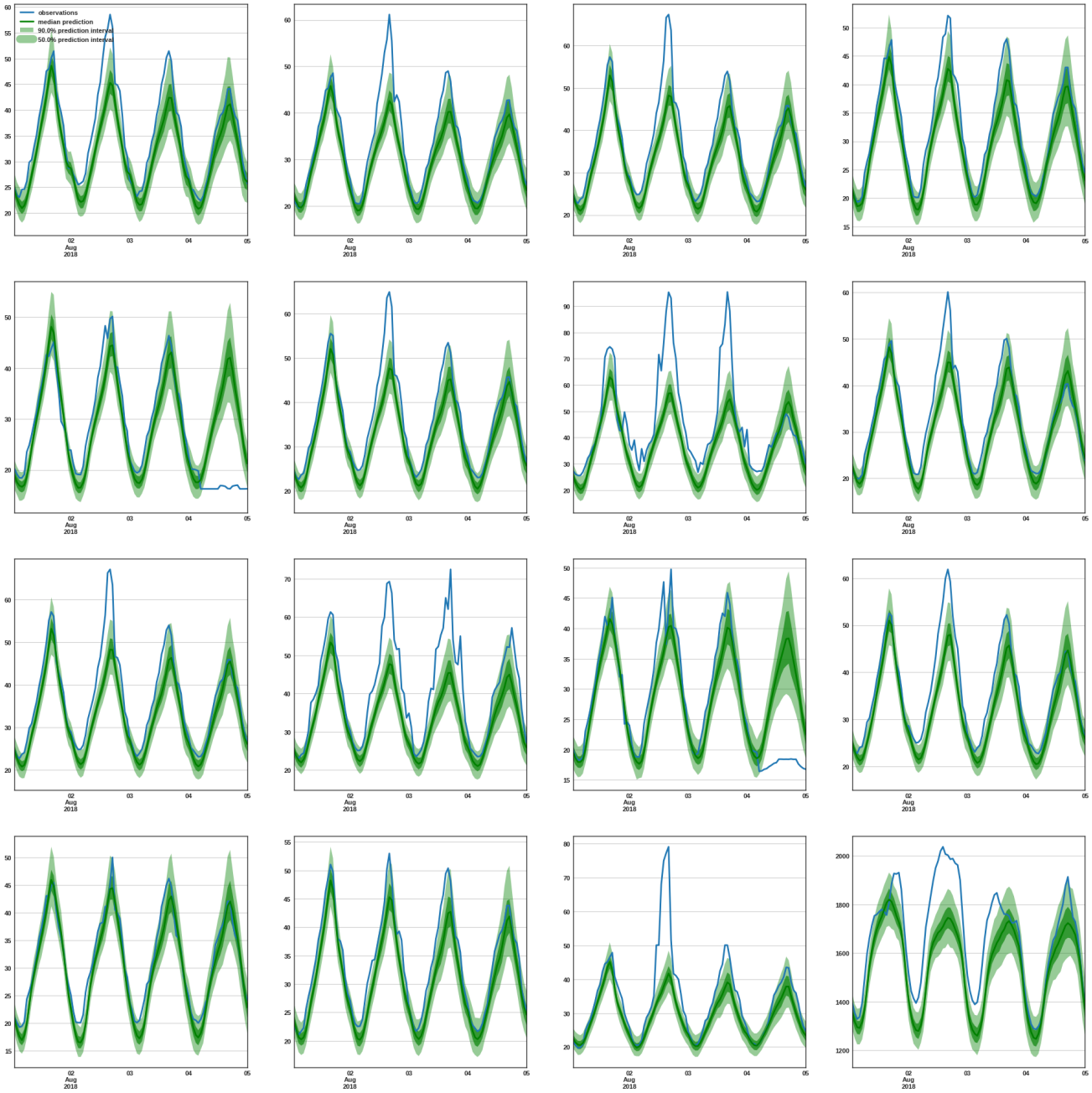


Figure 5.24: TimeGrad's hourly probabilistic forecasts for 4-days horizon.

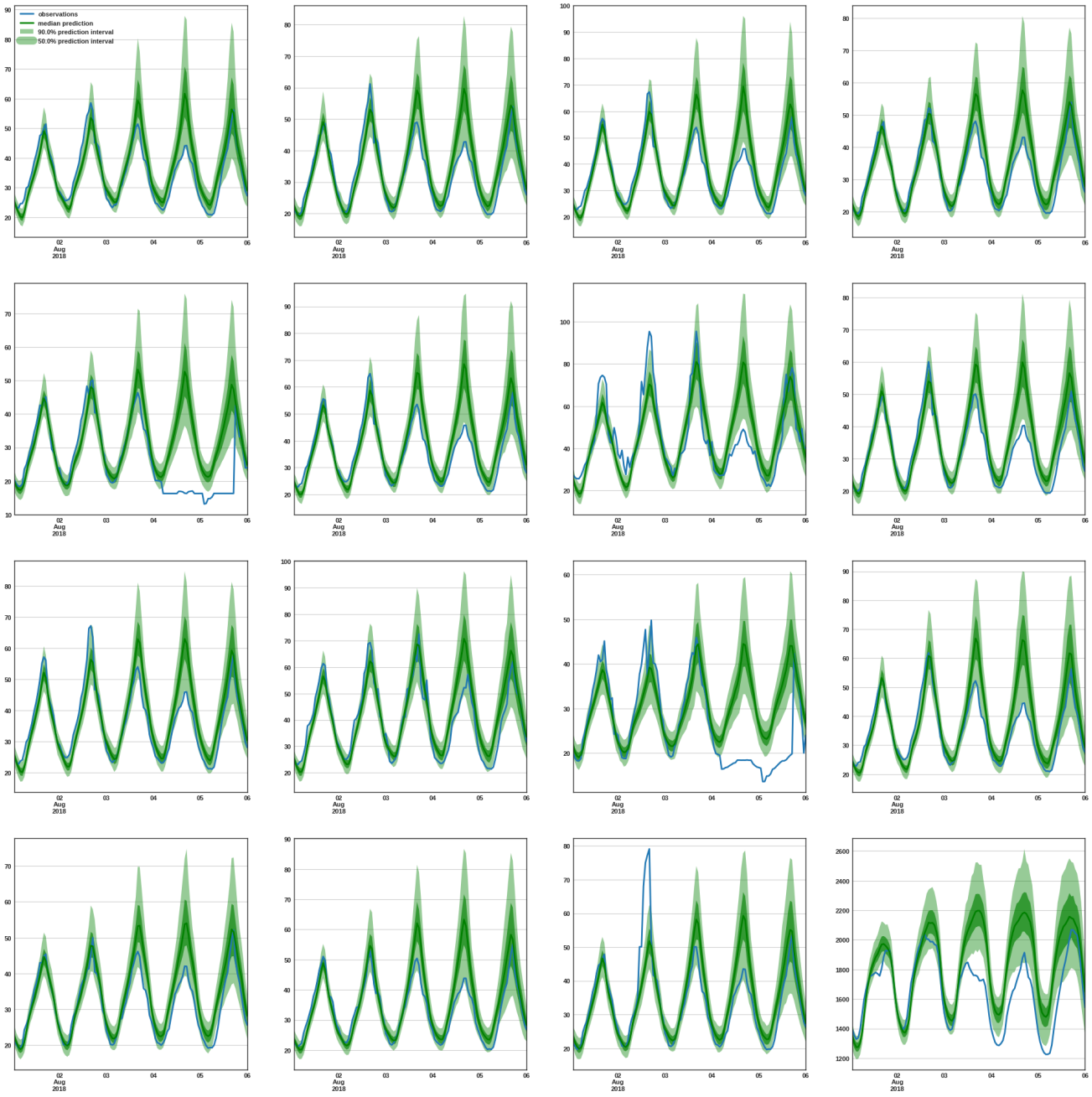


Figure 5.25: TimeGrad's hourly probabilistic forecasts for 5-days horizon.

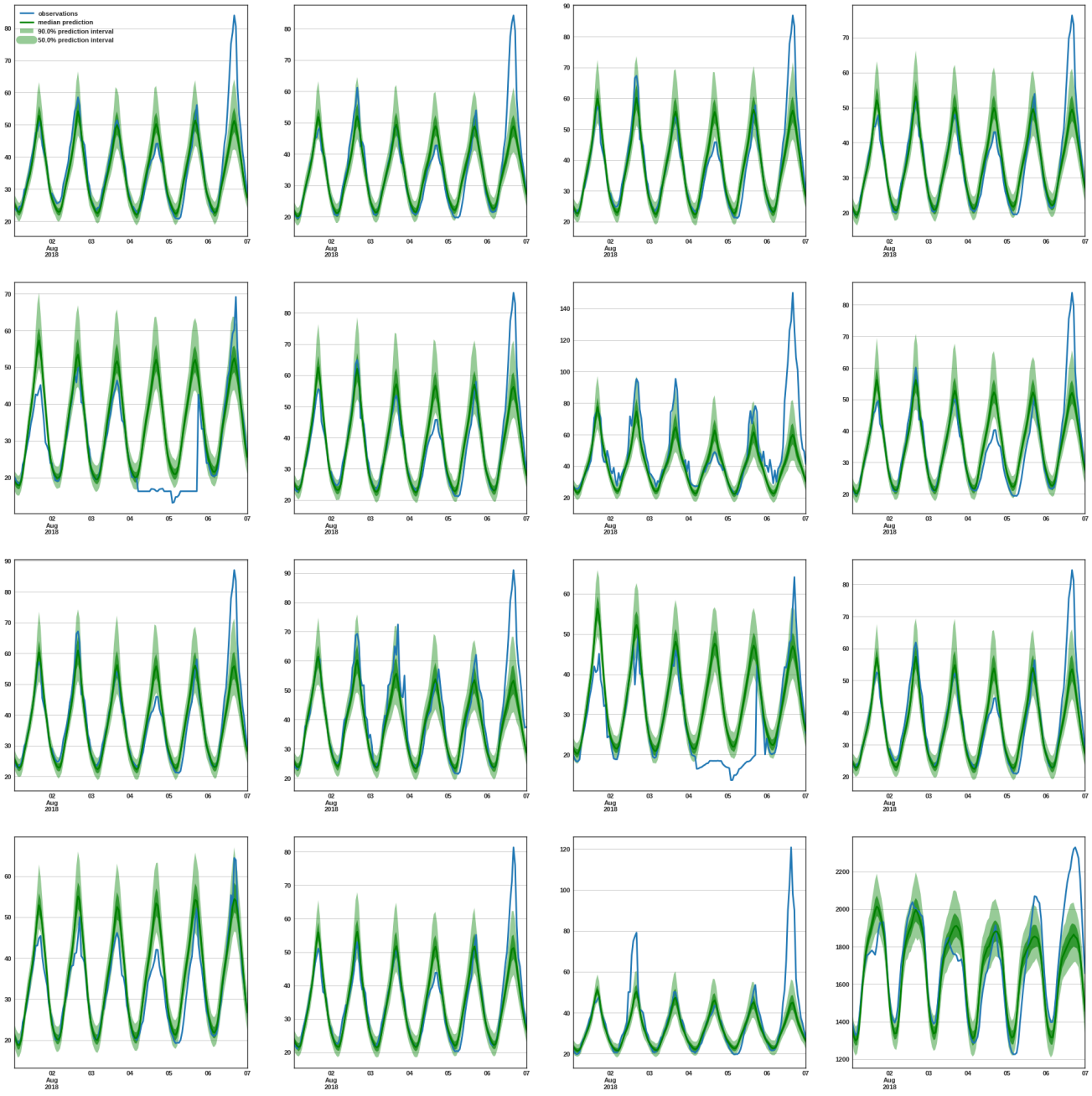


Figure 5.26: TimeGrad's hourly probabilistic forecasts for 6-days horizon.

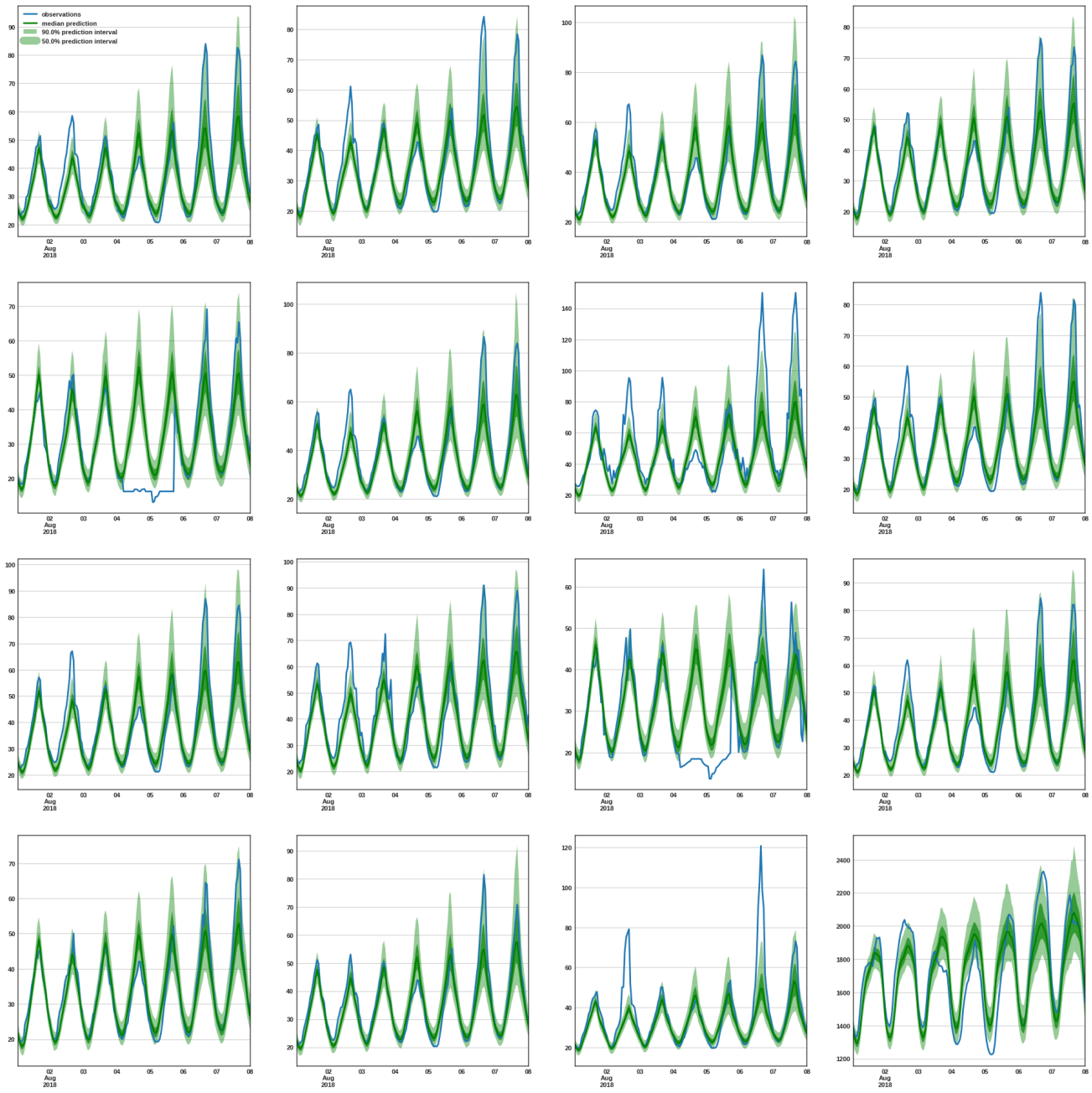


Figure 5.27: TimeGrad's hourly probabilistic forecasts for 7-days horizon.

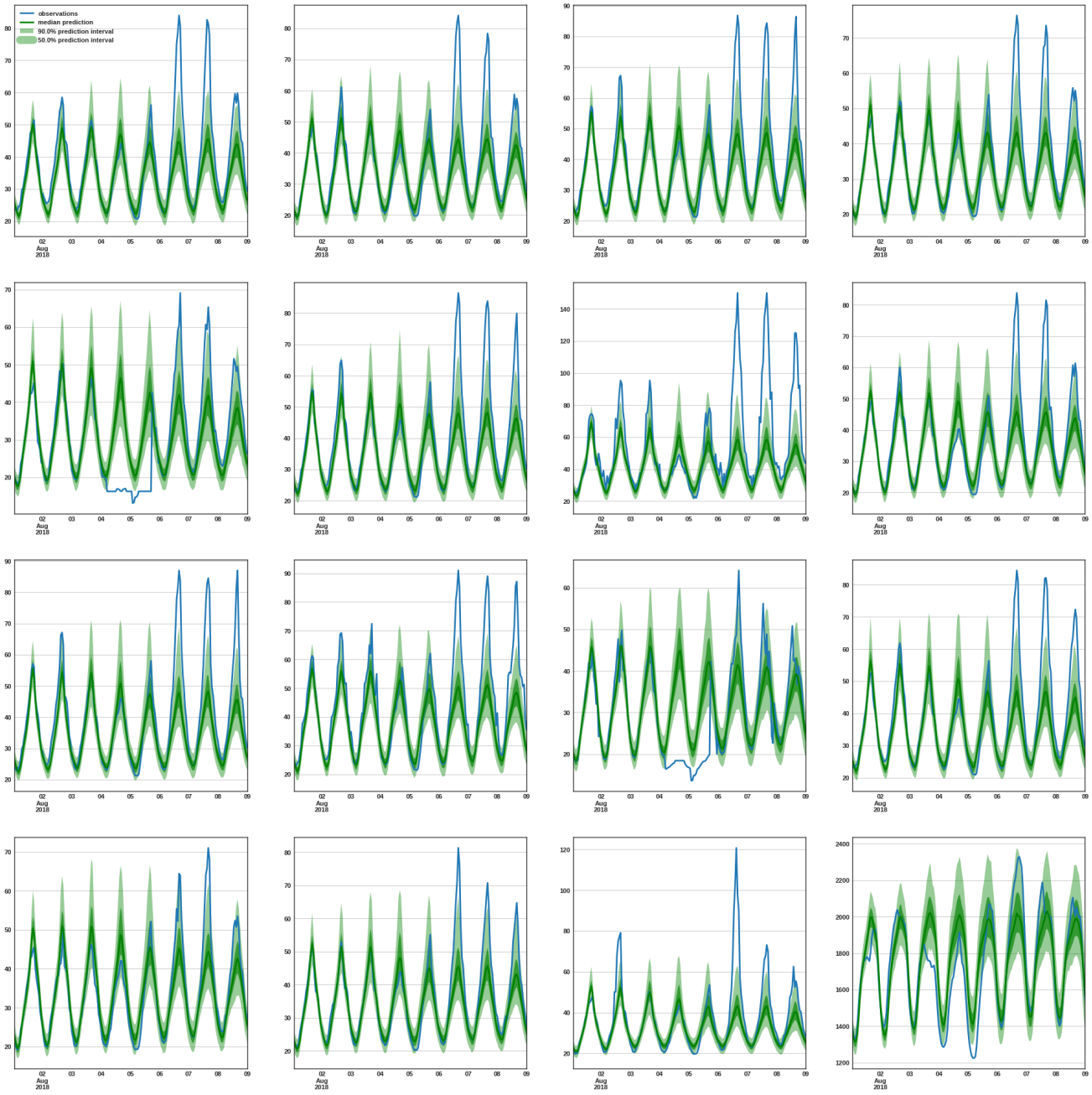


Figure 5.28: TimeGrad's hourly probabilistic forecasts for 8-days horizon.

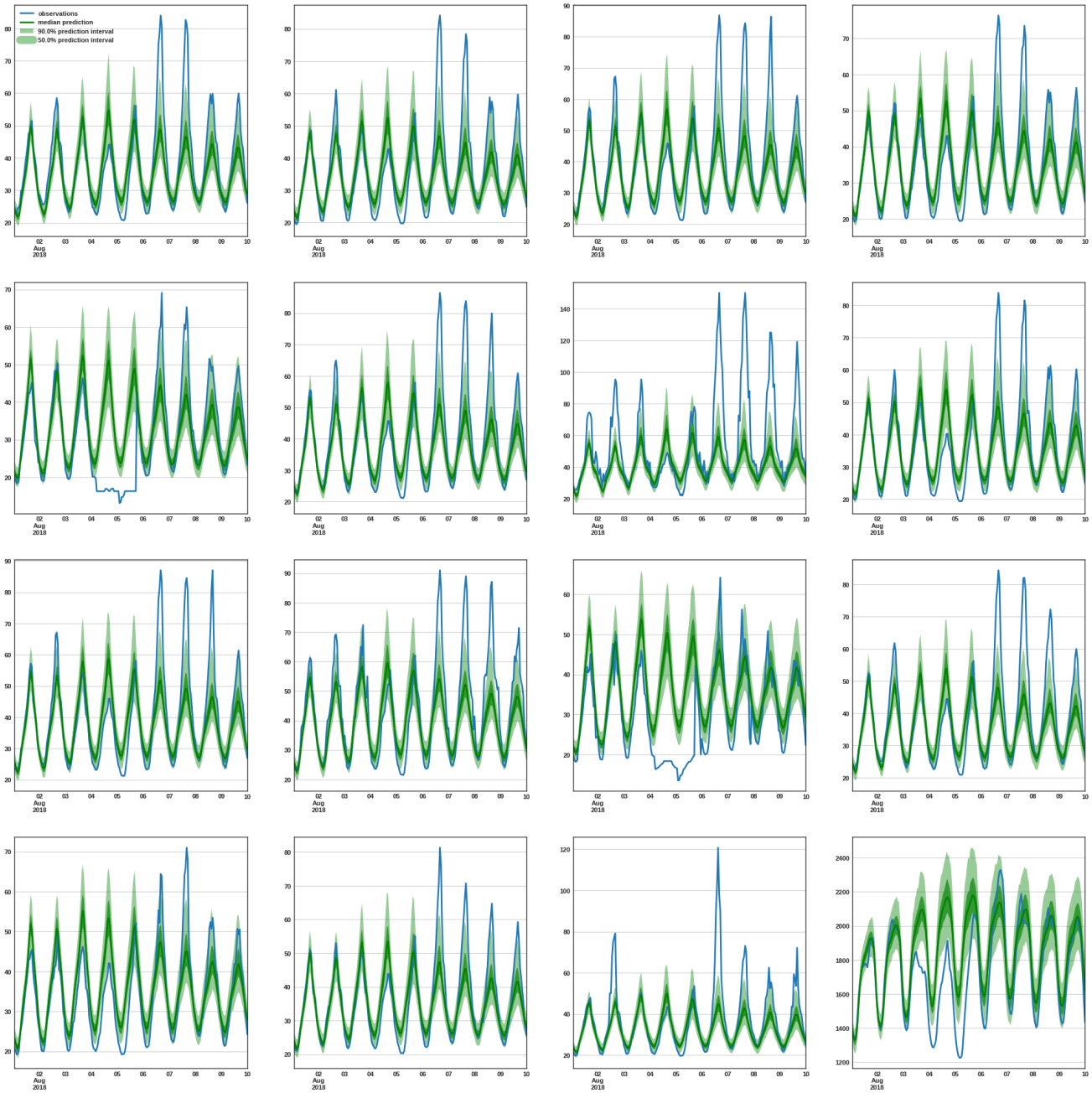


Figure 5.29: TimeGrad's hourly probabilistic forecasts for 9-days horizon.

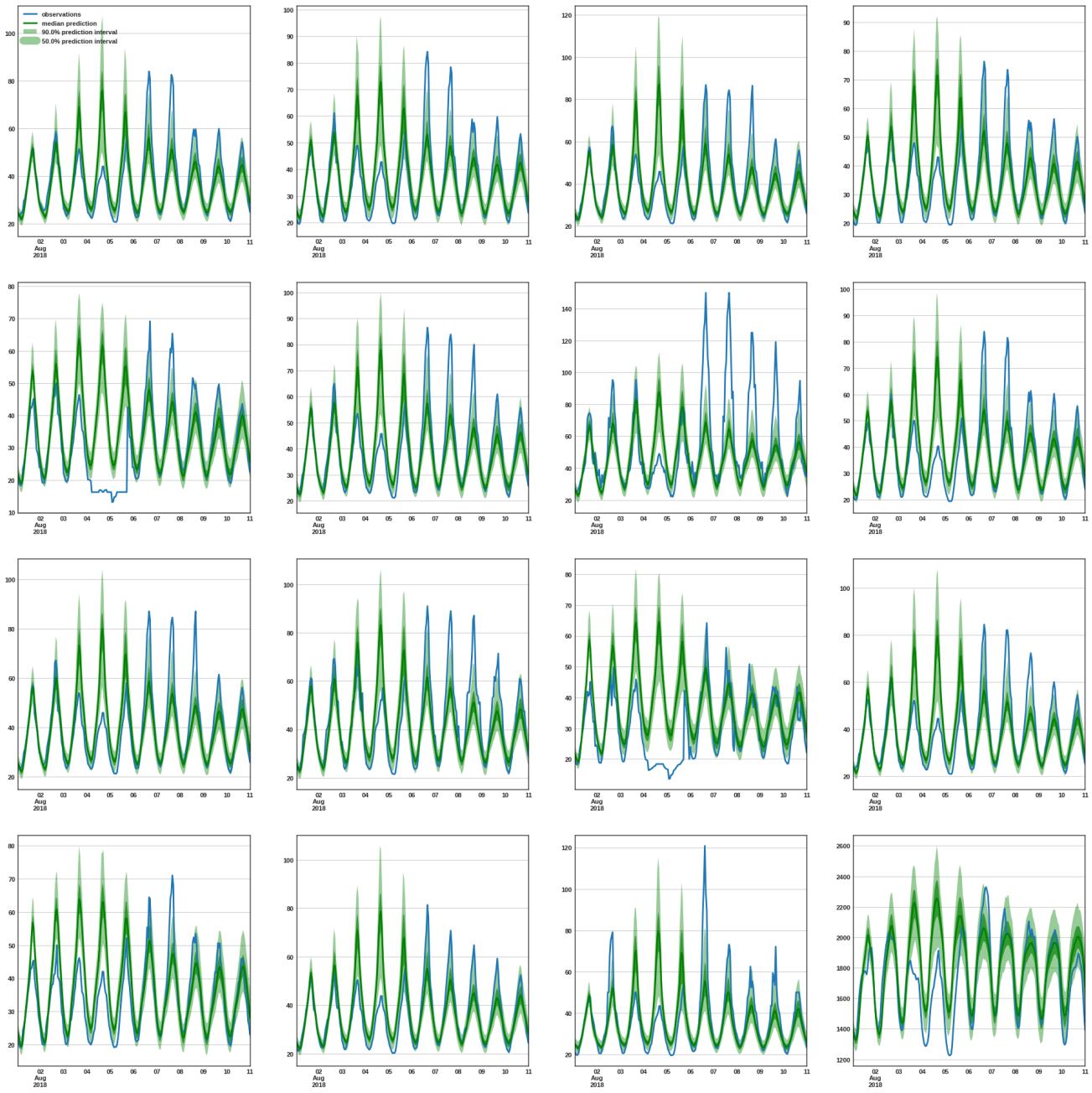


Figure 5.30: TimeGrad's hourly probabilistic forecasts for 10-days horizon.

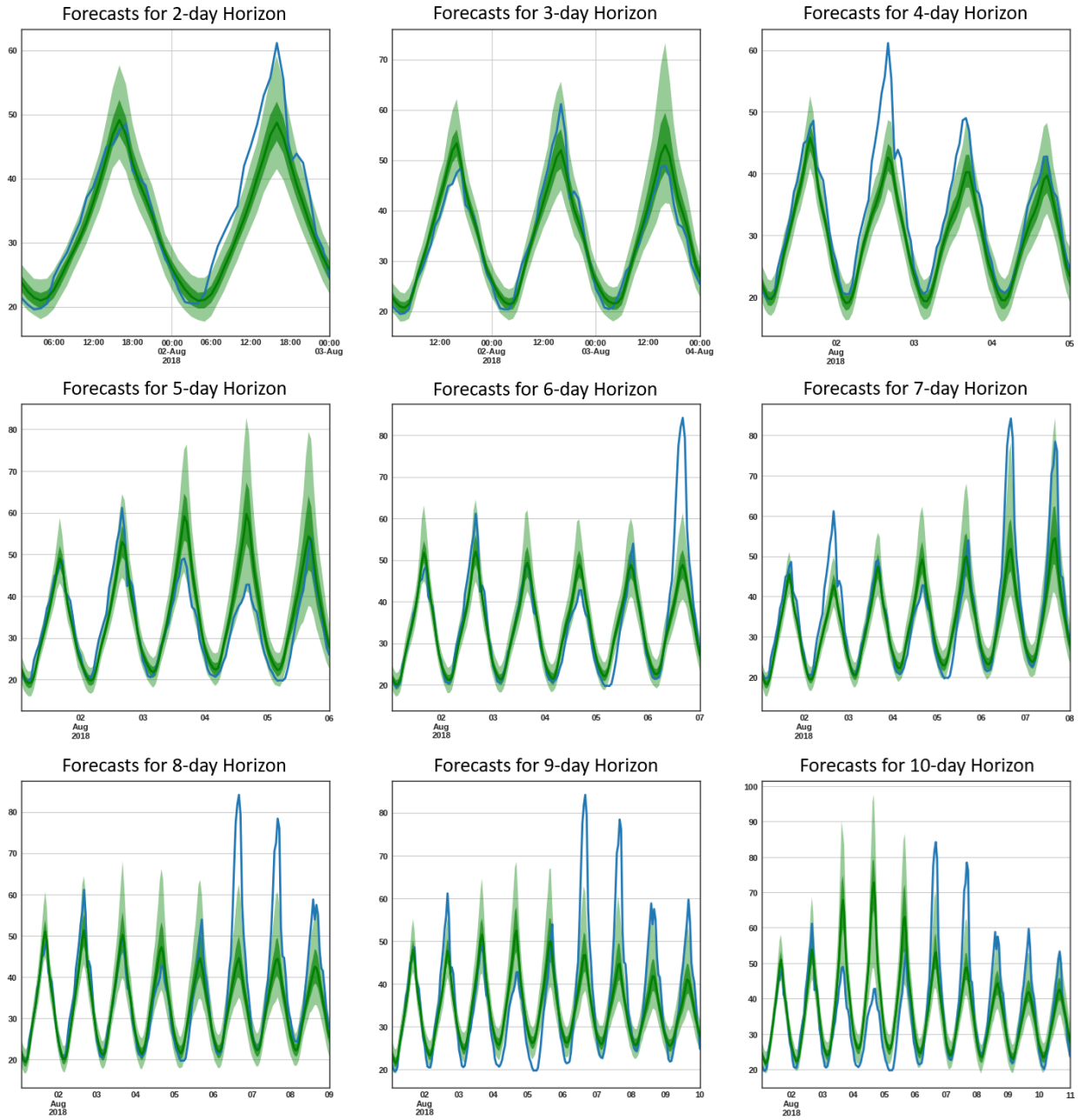


Figure 5.31: Comparison of TimeGrad's hourly probabilistic forecasts for a specific region for multiple horizons.

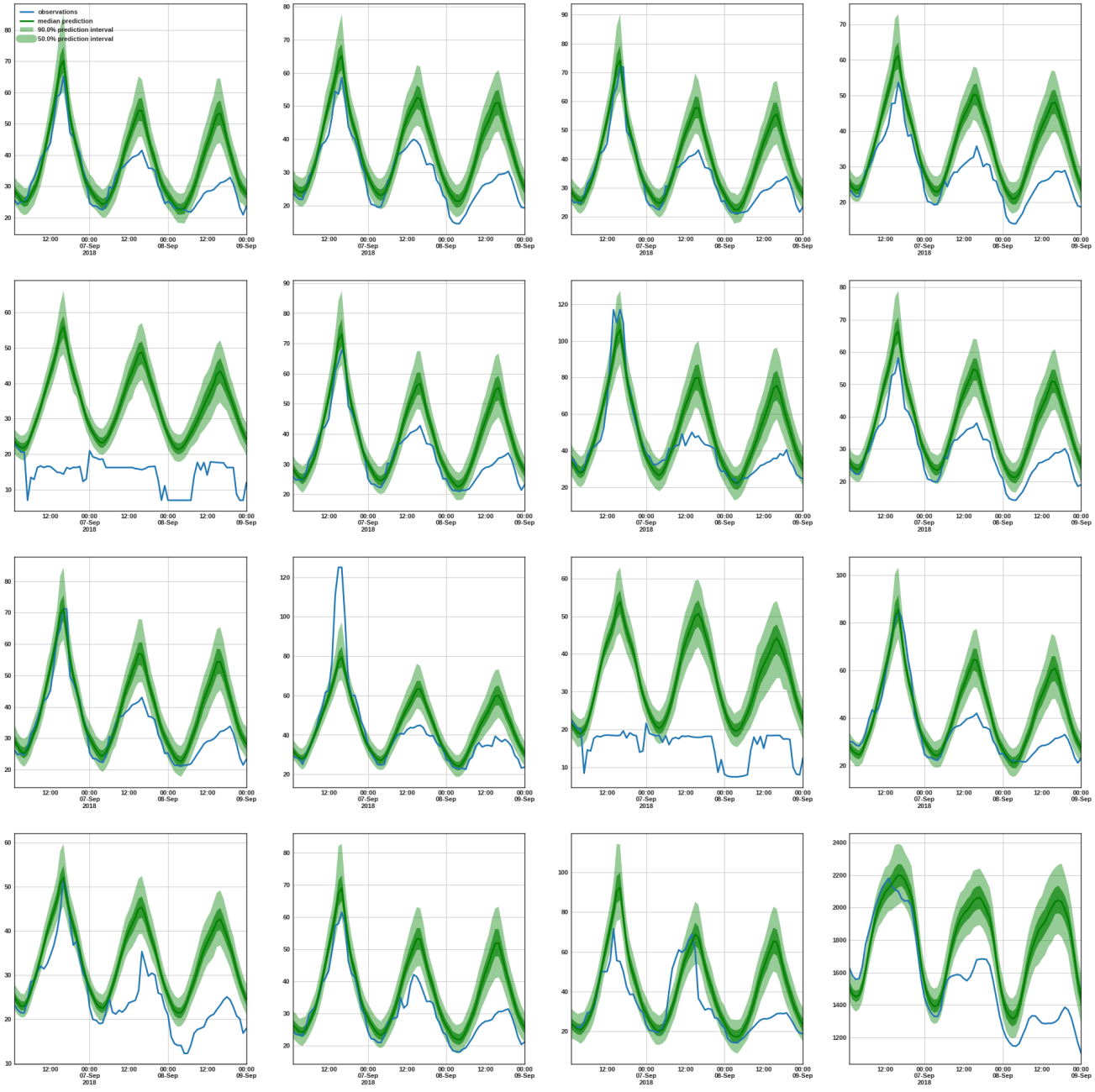


Figure 5.32: An example of TimeGrad's performance in time series anomalies.

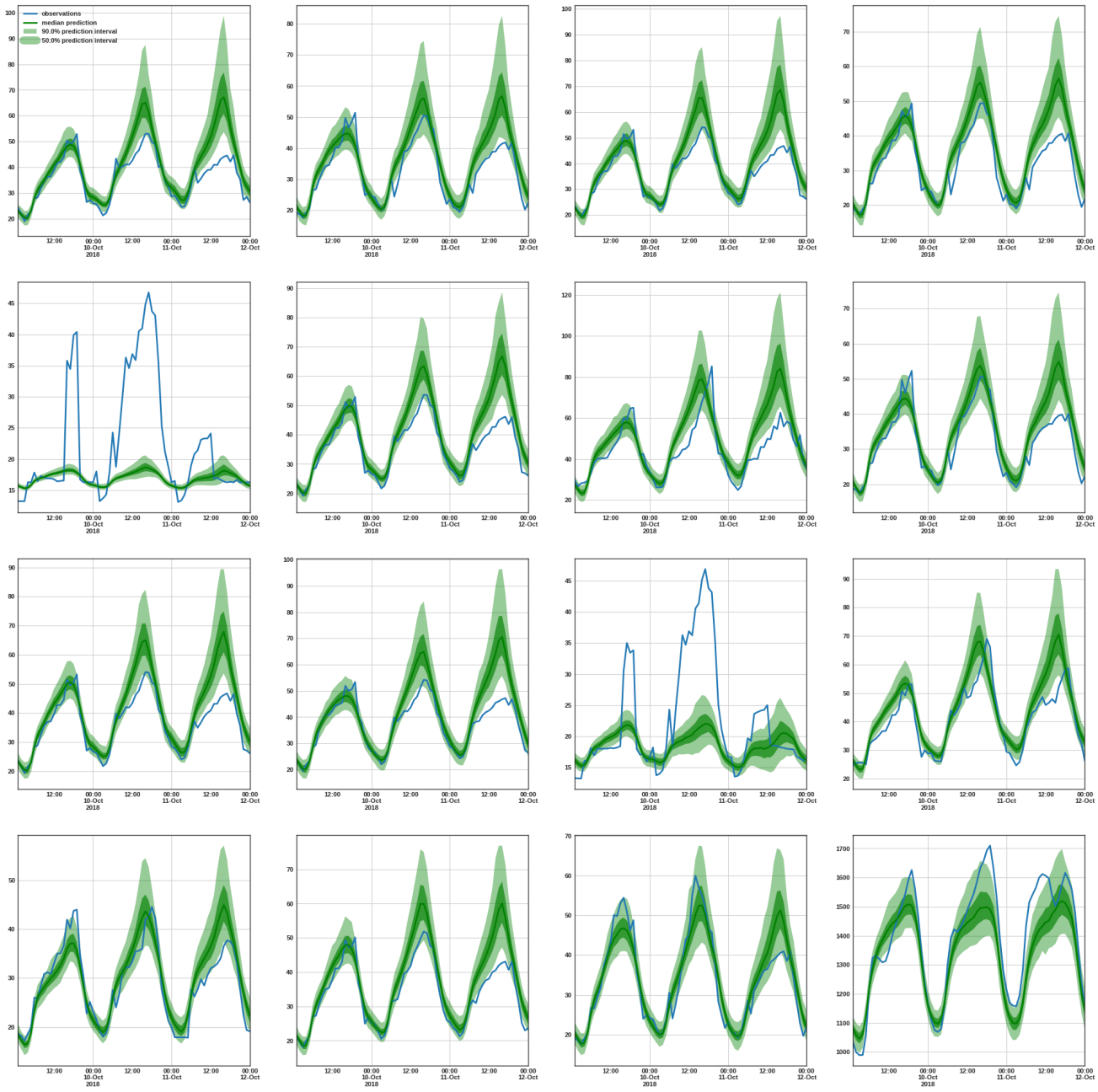


Figure 5.33: Another example of TimeGrad's performance in time series anomalies.

of unseen data, TimeGrad predicted the next 72 hours for each day across 27 time series. It is noteworthy that the LSTM within TimeGrad's diffusion process shares the exact same architecture as the standalone LSTM model, while yielding by far better results. This highlights the advantage of combining LSTM networks with the diffusion process. Typical examples of qualitative comparisons are shown in Figures 5.34 to 5.36, where TimeGrad's performance is better. Figure 5.37 illustrates an anomalous window of the time series, emphasizing TimeGrad's superior ability to model complex patterns. Furthermore, TimeGrad is trained on the full dataset of 27 time series, while the classical models AR, ARIMA, SARIMA, ETS are trained on one time series at a time, hence TimeGrad deals with a more complex problem. Despite this, TimeGrad performs better. Moreover, TimeGrad provides probabilistic predictions rather than point predictions, offering valuable additional information about the uncertainty of the forecasts.

Model	MSE	RMSE	MAE	MAPE	CRPS
AR	4.44×10^5	6.66×10^2	1.86×10^2	16.15	1.410×10^5
ARIMA	6.82×10^5	8.26×10^2	2.39×10^2	22.94	1.804×10^5
SARIMA	3.40×10^5	5.83×10^2	1.61×10^2	13.98	1.218×10^5
ETS	3.09×10^5	5.56×10^2	1.55×10^2	14.70	1.174×10^5
VAR	2.90×10^5	5.39×10^2	1.56×10^2	16.46	1.176×10^5
CNN	5.22×10^5	7.22×10^2	2.23×10^2	24.48	1.832×10^5
LSTM	4.07×10^5	6.38×10^2	1.82×10^2	17.17	1.372×10^5
TimeGrad	$\mathbf{1.97 \times 10^5}$	$\mathbf{4.44 \times 10^2}$	$\mathbf{1.31 \times 10^2}$	**11.46**	$\mathbf{7.635 \times 10^4}$

Table 5.3: Performance comparison of various models for time series forecasting.

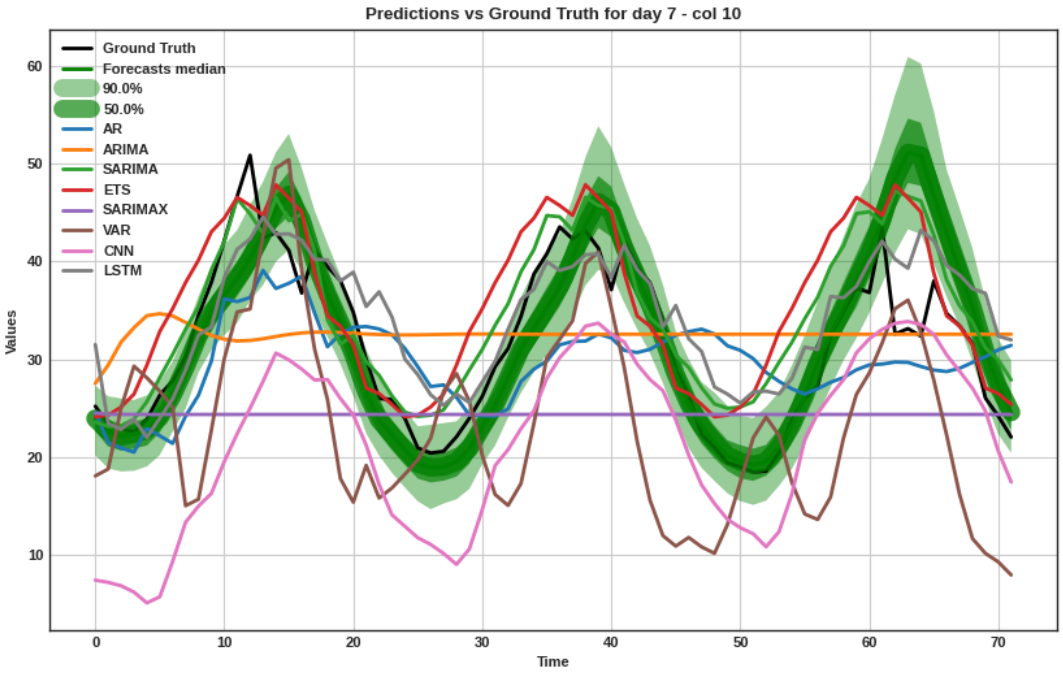


Figure 5.34: Comparison of TimeGrad's performance in 3-day prediction horizon with other classical methods and ground truth on a typical dataset window.

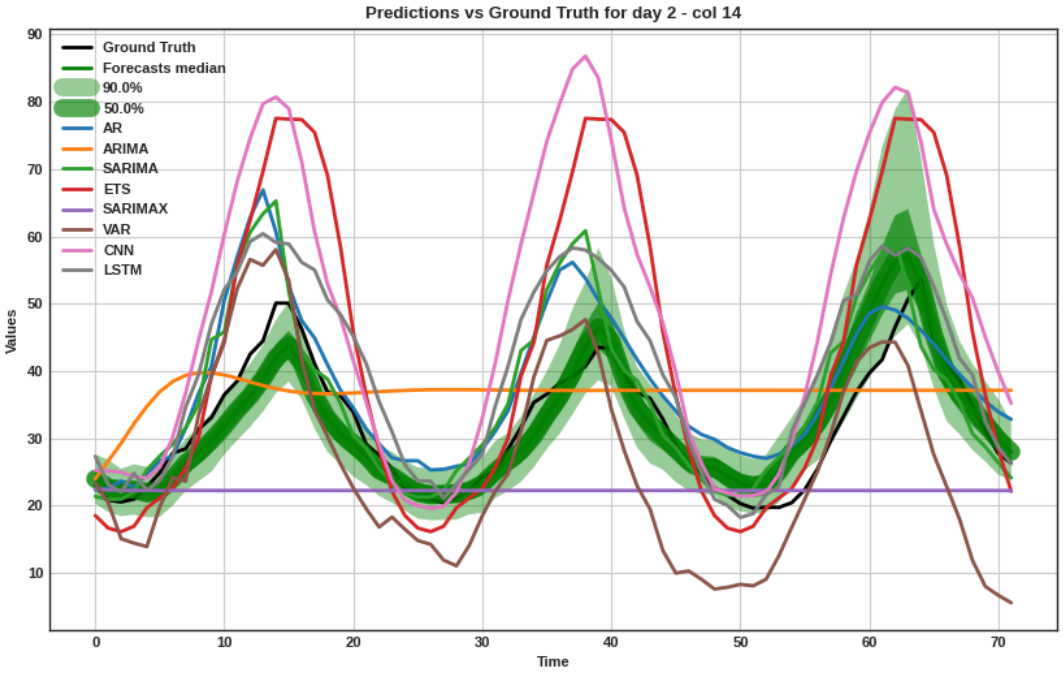


Figure 5.35: A second comparison of TimeGrad's performance in 3-day prediction horizon with other classical methods and ground truth on another dataset window.

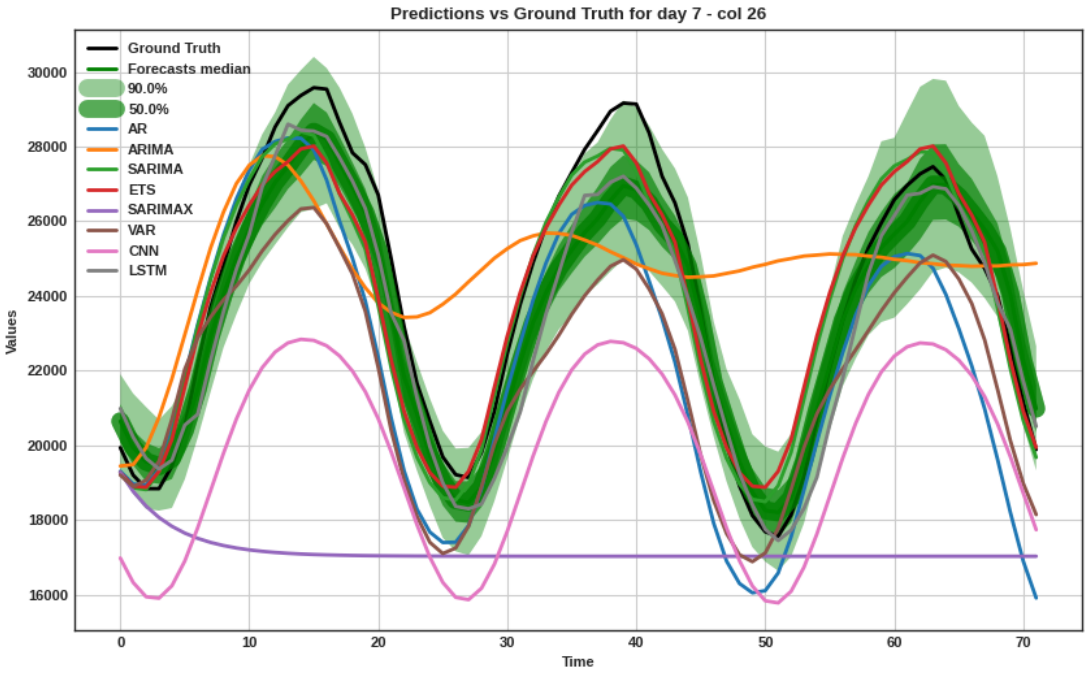


Figure 5.36: Comparison of TimeGrad's performance in 3-day prediction horizon with other classical methods and ground truth on a smooth segment of the dataset.

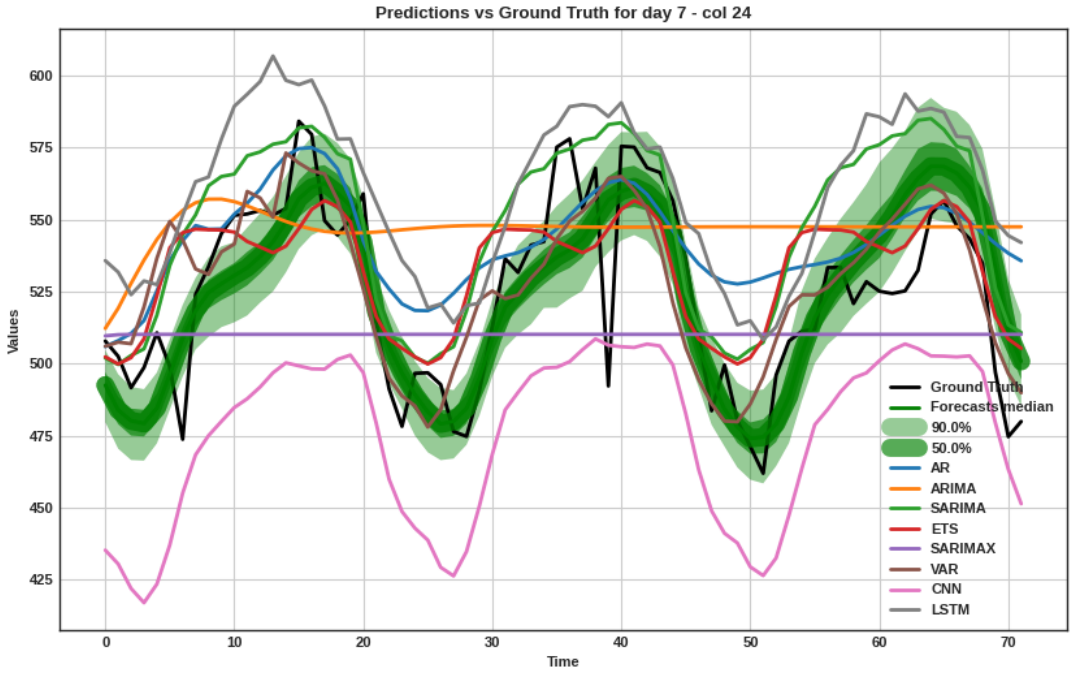


Figure 5.37: Comparison of TimeGrad's performance in 3-day prediction horizon with other classical methods and ground truth on a challenging dataset segment.

5.4 Environment

To incorporate the outputs of TimeGrad within the energy grid simulation, we defined an RL environment. This environment is structured to model the operation of a battery energy storage system (BESS) in the electricity grid as a Partially Observable Markov Decision Process. Forecast prices for a three-day horizon are generated using TimeGrad and are available in each day. The BESS characteristics include parameters such as the state of charge (SoC), efficiency, capacity, and degradation factors. These parameters model the operational constraints and physical characteristics of the battery system.

We model the operation of the BESS as a Partially Observable Markov Decision Process (POMDP)

$$(\mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma),$$

where:

- $\mathcal{S}$ is the (hidden) state space of the environment. In our application, the true state includes variables such as the battery's state-of-charge (SoC) and the electricity prices, which can be measured and other unobservable factors that influence price dynamics, which cannot be measured.
- $\mathcal{A}$ is the set of actions available to the agent. An action a_k is the amount of energy (expressed as a fraction of the battery capacity) to be charged (if $a_k > 0$) or discharged (if $a_k < 0$) at time step k.
- $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the state transition function; $T(s' \mid s, a)$ gives the probability of moving to state s' when action a is taken in state s.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, which captures the profit from energy transactions, accounting for the action costs (e.g., battery degradation due to excessive use). In this application, the agent can access the reward function through the observations, since it depends on the action and the observed market price, which are both observable.
- Ω is the set of observations available to the agent. Since the true state is not fully observable, the agent receives observations o_k that provide partial information about $\mathcal{S}$. In this application, the observations consist of the battery's SoC and the electricity prices of the various regions in the grid.
- $O : \mathcal{S} \times \mathcal{A} \to \Delta(\Omega)$ is the observation function that yields observations to the agent, after choosing an action at a given state. In our setting, the battery's SoC evolves deterministically according to the applied action, while the evolution of the electricity prices is stochastic.
- $\gamma \in [0, 1)$ is the discount factor of the POMDP.

At each time step k, the environment is in a hidden state s_k, and the agent receives an observation

$$o_k = \langle o_k^d,\, o_k^s \rangle,$$

where o_k^d is the deterministic component, which is the battery's SoC, evolving according to known dynamics and o_k^s is the stochastic component, consisting of the electricity prices, whose

dynamics are influenced by uncertain market factors. Based on the observed information, the agent selects an action $a_k \in \mathcal{A}$. The observation then transitions according to the dynamics:

$$\begin{aligned} \mathrm{SoC}_{k+1} &= \mathrm{SoC}_k + \eta\, a_k, \quad \text{with } \mathrm{SoC}_{k+1} \in [0,1], \\ p_{k+1} &= \mathrm{Prices}(k+1), \end{aligned}$$

where η denotes the battery efficiency when charging and discharging. The action space is defined by the battery's operational limits, i.e. the maximum flow of energy u_max that can flow in or out of the battery and the fact that the State of Charge should always lie between $[0, 1]$ (between 0% and 100% of the battery's capacity):

$$a_k \in [-u_{max}, u_{max}] \cap \left[\frac{SoC_k}{\eta}, \frac{1 - SoC_k}{\eta}\right] \Rightarrow a_k \in \left[-\min\left\{u_{max}, \frac{\mathrm{SoC}_k}{\eta}\right\}, \min\left\{u_{max}, \frac{1-\mathrm{SoC}_k}{\eta}\right\}\right].$$

The charging/discharging efficiency is set $\eta = 0.95$, which represents energy losses. We assume to have a battery with total energy capacity 192,000 Wh, and the maximum flow of energy in and out of the battery, which is bounded by the battery's physical limits, is $u_{max} = 0.25$ (or 25% of the capacity). To account for battery wear, the degradation of the BESS is modeled using a parameter $c_d = 4.5 \times 10^{-7}$ and the End-of-Life (EoL) threshold, which is set as $\mathrm{EoL} = 0.3$. The simulation operates with a time step of $\Delta t = 1$ hour.

So, the reward at each time step is computed as:

$$R(o_k, a_k) = \mathrm{Revenue}(o_k, a_k) - \mathrm{Degradation}(o_k, a_k),$$

where revenue is generated from buying or selling energy at the observed price and calculated as is calculated as $\mathrm{Revenue}(a_k, s_k) = a_k \cdot \eta \cdot p_k \cdot C \cdot 10^{-6}$, and the degradation cost is calculated as:

$$\begin{aligned} \mathrm{DoD} &= \frac{|SOC - SOC_{max}|}{SOC_{max}} \\ \text{Cycle Life} &= 0.0035 \cdot \mathrm{DoD}^3 + 0.2215 \cdot \mathrm{DoD}^2 - 132.29 \cdot \mathrm{DoD} + 10555 \\ \mathrm{Degradation} &= \begin{cases} c_d \cdot (1 - EoL) \cdot \Delta t \cdot \frac{SOC_{max}}{C} & \text{if } |a_k| = 0 \\ c_d \cdot (1 - EoL) \cdot \Delta t \cdot \frac{|a_k|}{2 \cdot \text{Cycle Life}} & \text{otherwise} \end{cases} \end{aligned}$$

where DoD is the Depth of Discharge, SoC is the State of Charge SoC_{max} is the maximum State of Charge (we assume that the battery can be fully charged, so $SoC_{max} = 1$), EoL is the End of Life of the battery, Δt is the time step, C is the battery capacity, a_k is the action chosen and c_d is the degradation cost coefficient. The degradation model was inspired by [41].

5.5 Model Predictive Control in Energy Arbitrage

In our first step, we integrate the diffusion-based forecasting model TimeGrad into the MPC framework defined in Section 4.4 to compute the optimal control sequence for managing the BESS in a partially observable environment. Let a decision epoch begin at step k_0. Denote by o_t^s the stochastic component of the observation at time t (i.e., the electricity price p_k) and by o_t^d the deterministic component (i.e., the battery SoC evolution). Given the history $\{o_0^s, \dots, o_{k_0-1}^s\}$, TimeGrad generates M sample forecasts:

$$\{\hat{o}_{k_0}^{s,(i)}\}_{i=1}^{M}.$$

We then aggregate these into a single point forecast using a desired aggregator (we experimented with both the median and the mean, which yielded similar results):

$$\hat{o}_{k_0}^s \;=\; \mathcal{F}\big(o_0^s, \dots, o_{k_0-1}^s\big) \;=\; \text{median}\big\{\hat{o}_{k_0}^{s,(i)}\big\}_{i=1}^{M},$$

The combined predicted next-step observation is therefore:

$$\hat{o}_{k_0} \;=\; \big\langle o_{k_0}^d,\; \hat{o}_{k_0}^s \big\rangle.$$

Using this one-step forecast operator in an autoregressive fashion, we generate a predicted observation trajectory $\{\hat{o}_k\}_{k=k_0}^{k_0+N}$. The resulting deterministic MPC problem, over a horizon of N steps, becomes

$$\begin{aligned}
\underbrace{\text{maximize}}_{a_{k_0},\dots,a_{k_0+N-1}} \quad & \sum_{k=k_0}^{k_0+N-1} R\big(\hat{o}_k,\; a_k\big) \\
\text{s.t.} \quad & \hat{o}_{k_0-1} \;=\; o_{k_0-1}, \quad \hat{o}_k = \big\langle f(\hat{o}_{k-1}^d, a_{k-1}),\; \hat{o}_k^s \big\rangle, \\
& \hat{o}_k^s \;=\; \mathcal{F}\big(o_0^s, \dots, o_{k-1}^s, \hat{o}_{k_0}^s, \dots, \hat{o}_{k-1}^s\big), \\
& a_k \in \mathcal{A}, \quad k = k_0, \dots, k_0 + N - 1.
\end{aligned}$$

Here f denotes the known, deterministic SoC update $o_{k+1}^d = f(o_k^d, a_k)$, and $R(\hat{o}_k, a_k)$ is the reward (revenue minus the degradation cost) based on the predicted observation. By aggregating TimeGrad's uncertainty into a single forecast at each step, the optimization remains deterministic. In contrast, the stochastic MPC framework that we examine in the next section explicitly includes multiple sampled trajectories inside the optimizer, to handle the uncertainty in the optimization step.

At each day t, we solve the optimization problem over a rolling 24-hour horizon using the latest TimeGrad forecasts. Only the first 24 actions are applied; once NYISO publishes the real prices, we observe the new state, shift the horizon forward one day, and re-optimize.

Figures 5.38 to 5.42 show the control strategies that the MPC plans over five consecutive days in the market. The agent plans using the forecasts of the next three days. Once the optimal strategy is calculated, the agent performs the best actions for the first day and moves to the next day, when the prices of the previous day are known, since they were set by NYISO after collecting all the bids, and more accurate forecasts can be generated.

Figures 5.43 and 5.44 show the control sequence of the BESS that the MPC calculated and the SoC along with the forecasted prices and actual prices after publication, over a month. Specifically, for Figure 5.44, the blue line refers to the SoC of the BESS for each time step, the gray dotted line is the region's forecasted prices for each time step and the black line represents

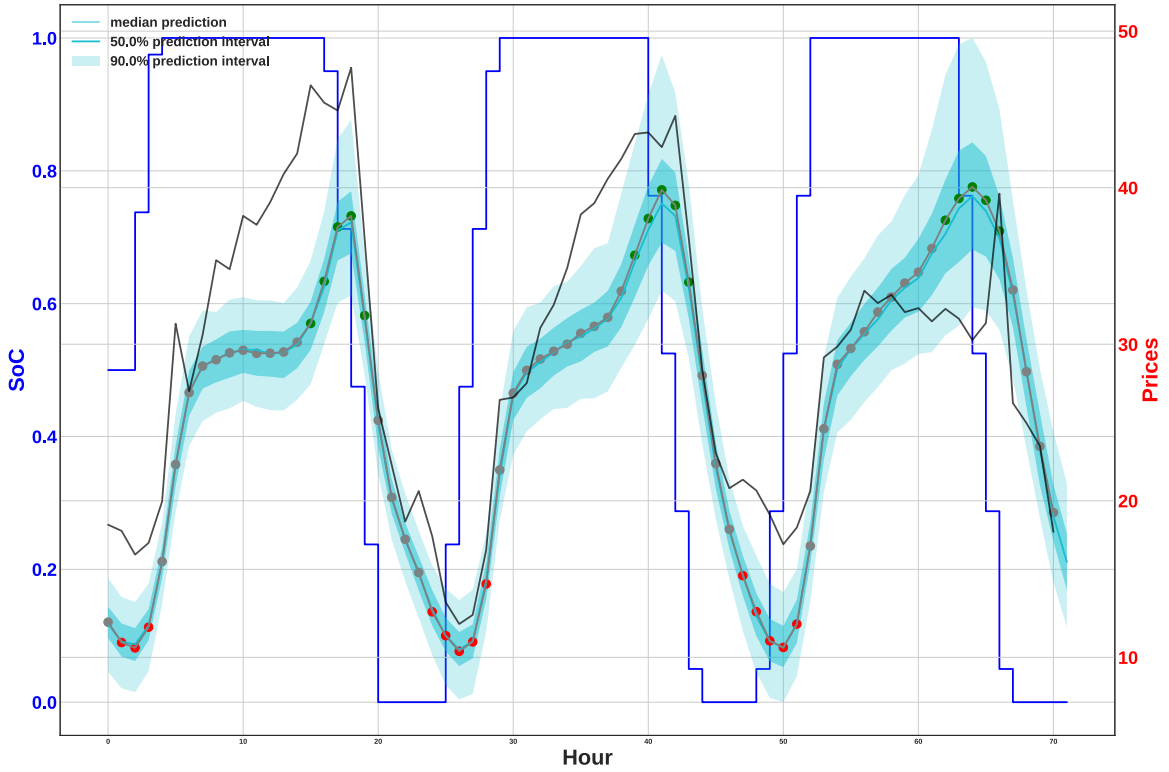


Figure 5.38: First day's strategy planned by the MPC Optimizer.

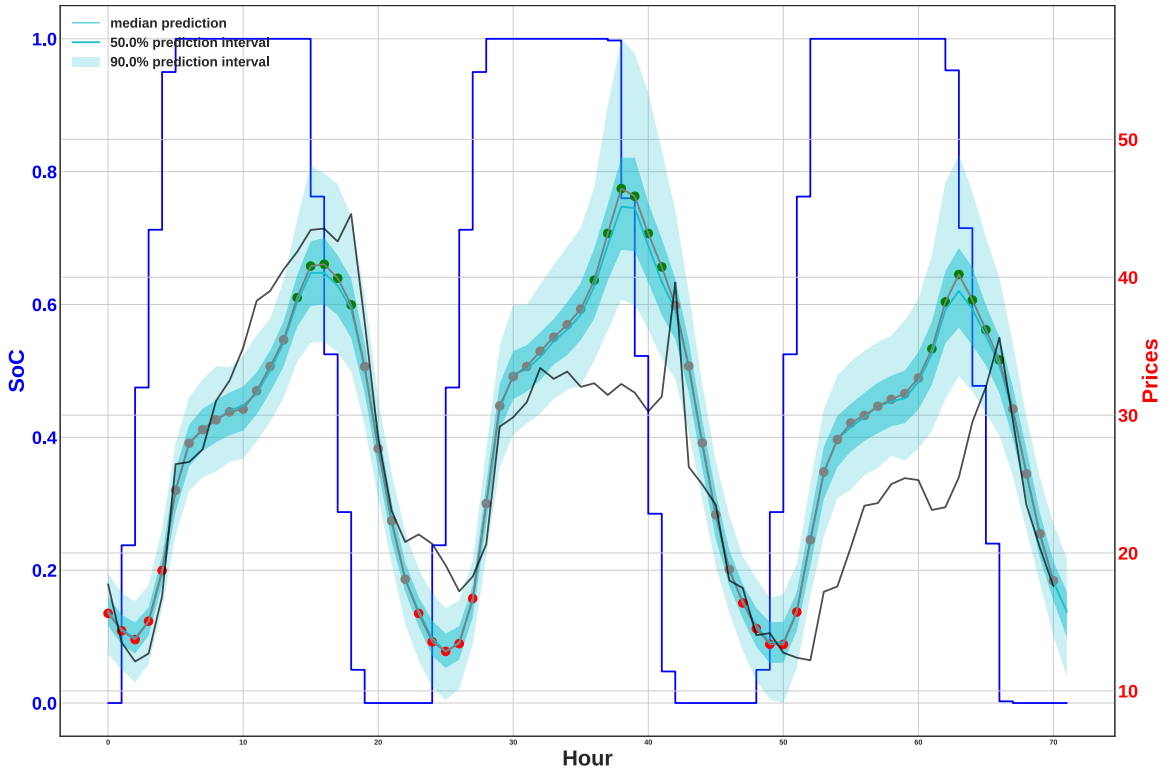


Figure 5.39: Second day's strategy planned by the MPC Optimizer, after new knowledge occurred (new prices published by NYISO).

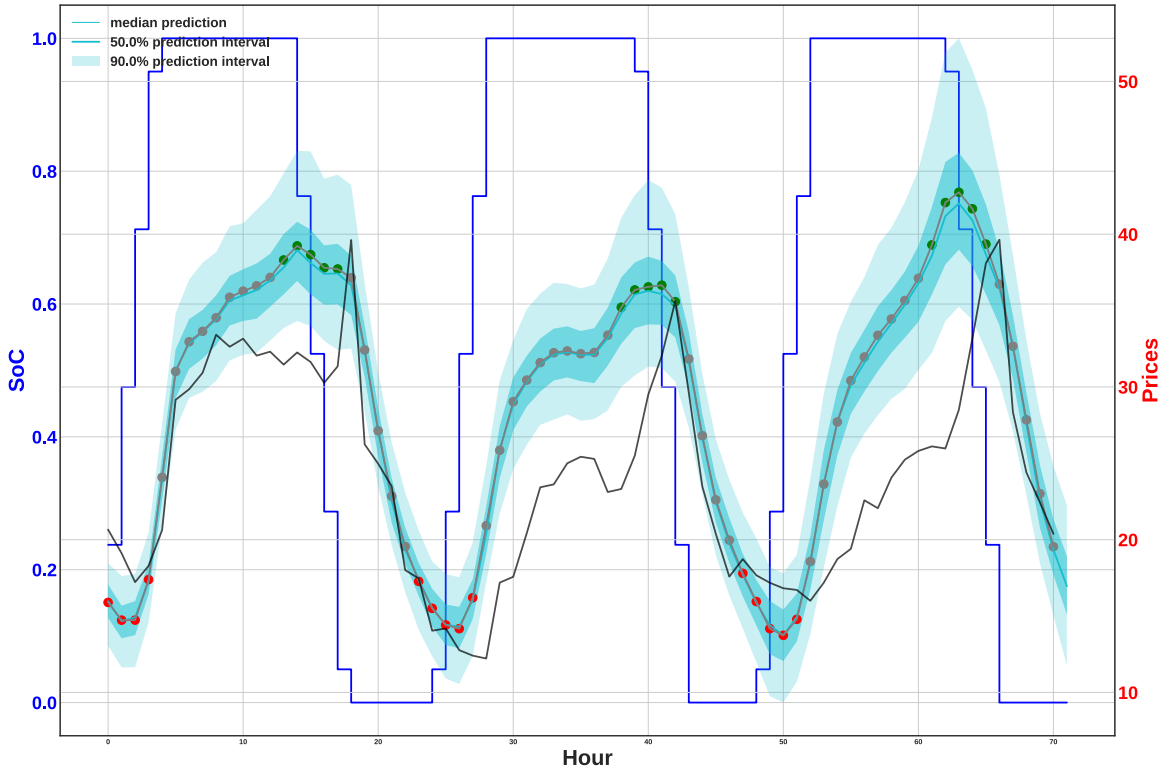


Figure 5.40: Third day's strategy planned by the MPC Optimizer, after new knowledge occurred (new prices published by NYISO).

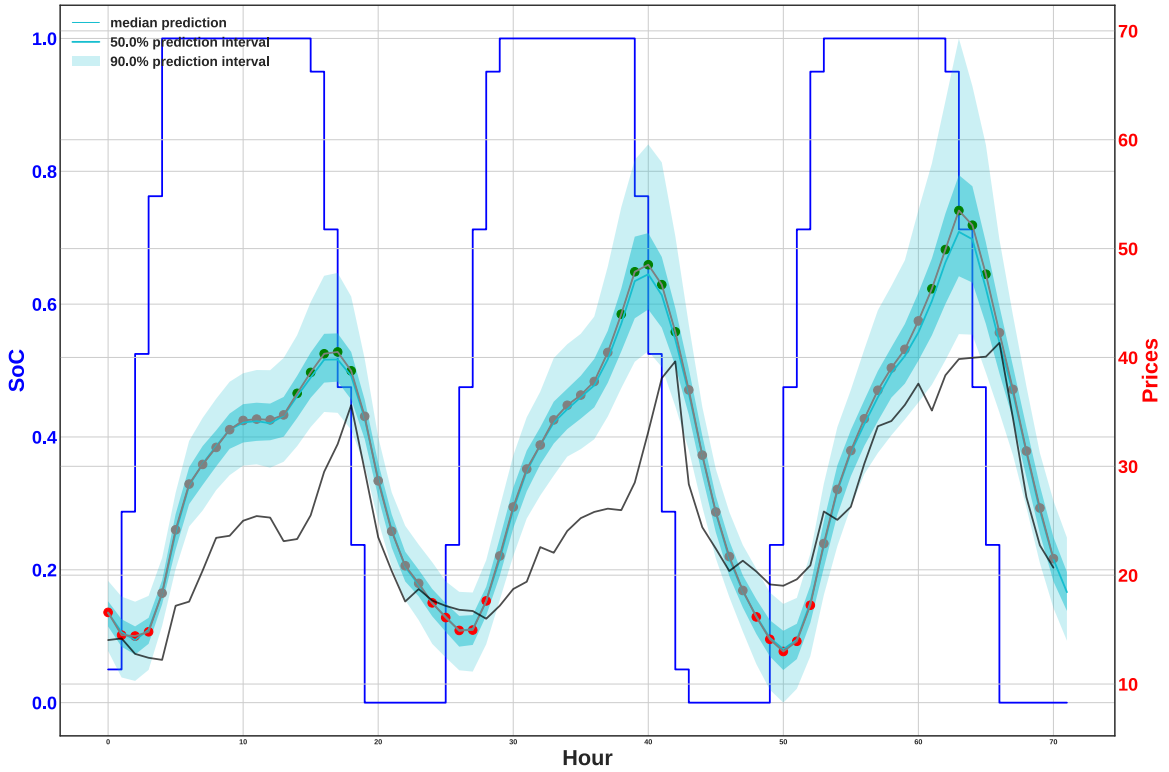


Figure 5.41: Fourth day's strategy planned by the MPC Optimizer, after new knowledge occurred (new prices published by NYISO).

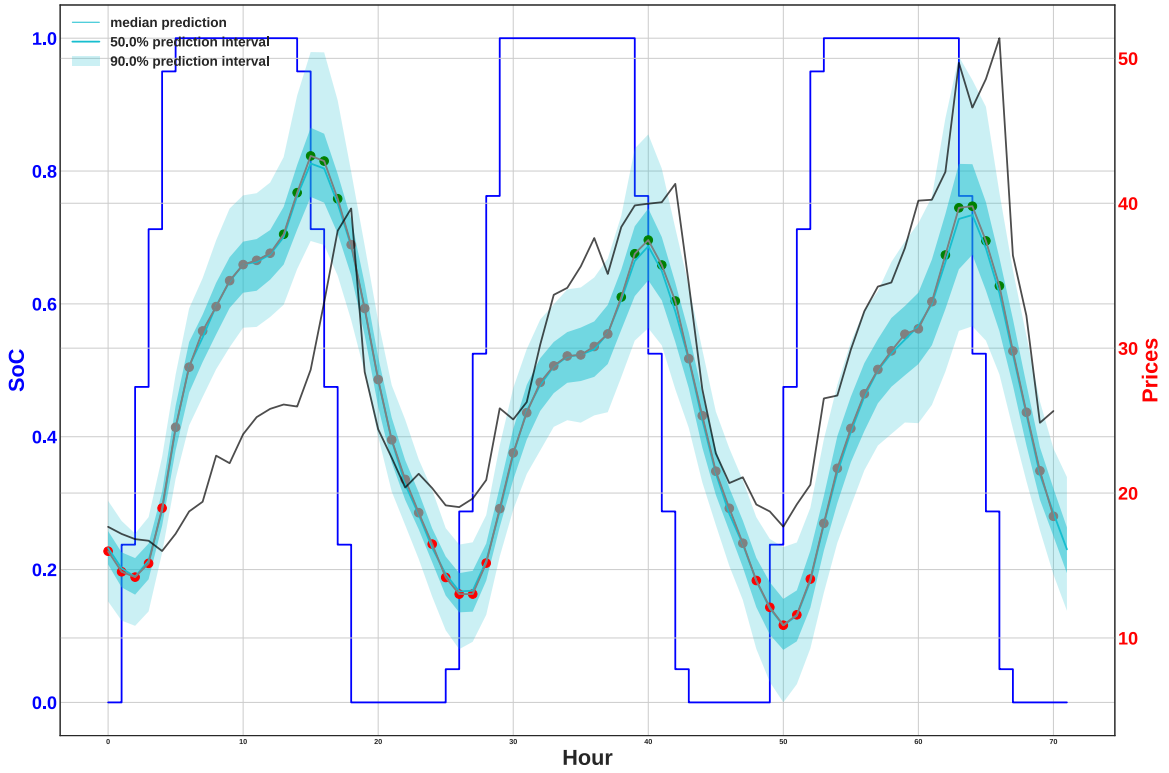


Figure 5.42: Fifth day's strategy planned by the MPC Optimizer, after new knowledge occurred (new prices published by NYISO).

the actual prices. The color of the dots on the gray line helps understand the action. If the SoC increases, it means that the BESS buys energy from the grid, thus the revenue is negative, and the dot's color is red. If the SoC remains the same, it means that the BESS makes no action on the grid, so there is no revenue (only maintenance costs to retain the charge). In this case, the dot's color is gray. And if the SoC decreases, it means that the BESS sells energy to the grid, thus the revenue is positive, and the dot's color is green.

Overall, the BESS operates efficiently, typically buying energy when prices are low and selling when prices are high. It also strategically charges the BESS when it anticipates prices to increase and discharges when prices peak to maximize profit.

One important aspect to note is that the BESS is limited in how much energy it can charge or discharge at any given time due to its inherent characteristics. As a result, it cannot fully charge or discharge on the most favorable prices in just a single time step. Instead, it must spread out charging and discharging across multiple steps. The system charges more when prices are optimal and gradually adjusts the flow of energy in less favorable time periods to either fill or empty the BESS.

A significant limitation of the MPC algorithm in the BESS setup is that it optimizes over a finite horizon. As seen in each of the Figures 5.38 to 5.42, by the end of the horizons, the plan suggests that the battery is fully discharged to maximize the cumulative reward over the horizon. While this is optimal short-term, it leaves the BESS depleted, without taking into account the future energy needs and opportunities. MPC is efficient at optimizing short-term profits but lacks long-term planning capabilities. In later sections, we will examine how the proposed solution to this problem in Section 4.6 manages to address this problem by aiming for

a terminal SoC that is expected to be optimal for future time steps.

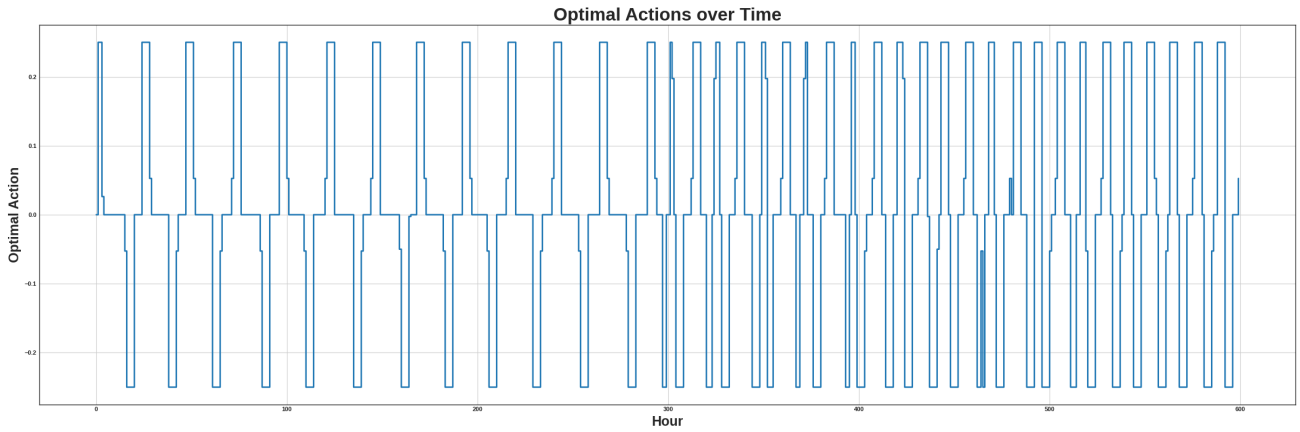


Figure 5.43: Optimal actions returned for 25 days by the MPC optimizer.

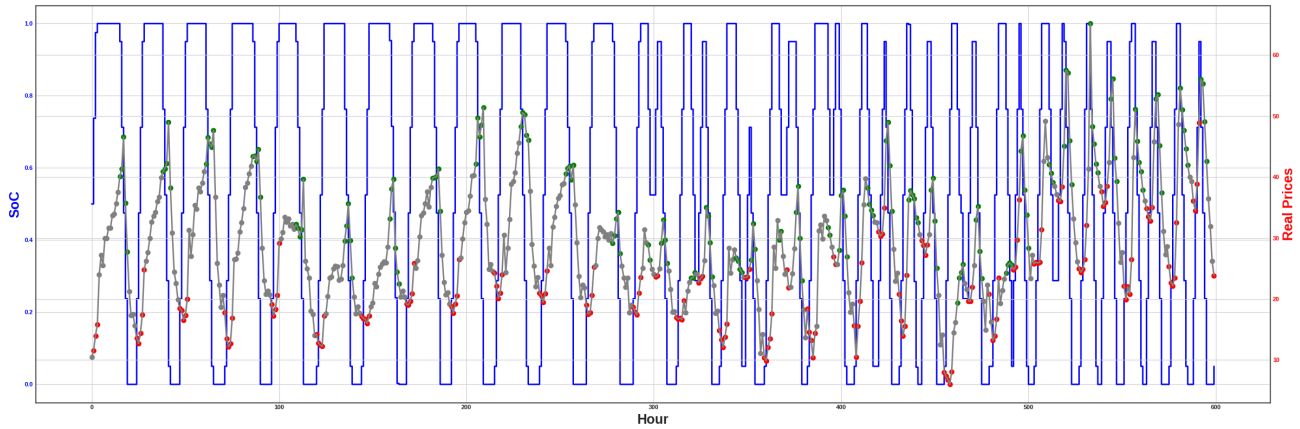


Figure 5.44: SoC levels along with fixed hourly prices for 25 days after applying the MPC Optimizer.

5.6 Stochastic MPC in Energy Arbitrage

Stochastic Model Predictive Control (SMPC) extends the classical MPC framework by embedding uncertainty directly into the optimization. Unlike traditional MPC, which relies on deterministic forecasts, SMPC accounts for the stochastic nature of variables such as future energy prices by considering probabilistic forecasts. As a first step toward full Monte Carlo SMPC, we follow the framework for Monte Carlo SMPC, defined in Section 4.5.1, with number of generated samples $M = 1$, by drawing a single stochastic price trajectory from TimeGrad's forecast distribution. At decision epoch k_0, we sample

$$\hat{o}^s_{k+1} \sim p_\theta\left(\hat{o}^{s,(i)}_{k+1} | \mathbf{h}^{(i)}_k\right), \qquad k = k_0, \ldots, k_0 + N - 1,$$

where $\mathbf{h}_k$ is TimeGrad's RNN state. Each sampled stochastic component is combined with the known deterministic update

$$o^d_{k+1} = f\big(o^d_k,\, a_k\big)$$

to form the full predicted observation

$$\hat{o}_{k+1} \;=\; \big\langle o^d_{k+1},\; \hat{o}^s_{k+1} \big\rangle.$$

The controller then maximizes the cumulative reward along this single trajectory:

$$\mathcal{J} \;=\; \sum_{k=k_0}^{k_0+N-1} R\big(\hat{o}_k,\, a_k\big), \quad \text{where } \hat{o}_k = \big\langle o^d_k,\; \hat{o}^s_k \big\rangle.$$

SMPC enables the system to plan control actions over the prediction horizon, taking into account one realization of the stochastic forecasts of future prices. However, taking only one trajectory often results in suboptimal control actions, since it will most likely not be a good representative sample. The stochasticity can lead to frequent charging and discharging decisions as the controller tries to exploit opportunities for profit, that may not be realistic. Figures 5.45 to 5.49 show the control strategies that the SMPC plans over five consecutive days in the market. The agent plans using the sampled trajectory of the next three days. Again, once the optimal strategy is calculated, the agent performs the best actions for the first 24 hours and moves to the next day, when the prices for the previous day are published and more accurate forecasts can be generated.

The bold blue line refers to the SoC of the BESS for each time step, the gray dotted line is the mean of the forecasted prices for each time step, while the purple line is the sample used for the optimization and the black line refers to the actual prices as published by NYISO.

By looking at the dots' colors and the thin purple line in the Figures 5.45 to 5.49 it is easy to understand that the BESS operates less efficiently than in the case of deterministic MPC. Although again, in general, it buys energy when prices are low and sells when prices are high and it charges the BESS when it anticipates prices to increase and discharges when prices peak. The noisy trajectory results in the algorithm yielding suboptimal control sequence. This can be seen in the daily plans plots that the agent plans to charge and discharge the battery very often, as the noise makes the agent misinterpret it as profit opportunity.

Despite these challenges, SMPC produces a logical sequence of actions, because TimeGrad is a powerful model and even taking only one sample of the distribution yields a good action strategy. This is important as there can be applications were the generation of multiple trajectories is not

possible, due to time or computational bounds, therefore having a model that can perform well even if it is not fully exploited, is crucial. In the next implementation we incorporate Monte Carlo simulations, to enhance the robustness of SMPC.

Moreover, like deterministic MPC, SMPC optimizes over a finite horizon and may prioritize short-term gains at the expense of long-term considerations, as seen in the tendency to fully discharge the battery by the end of the horizon.

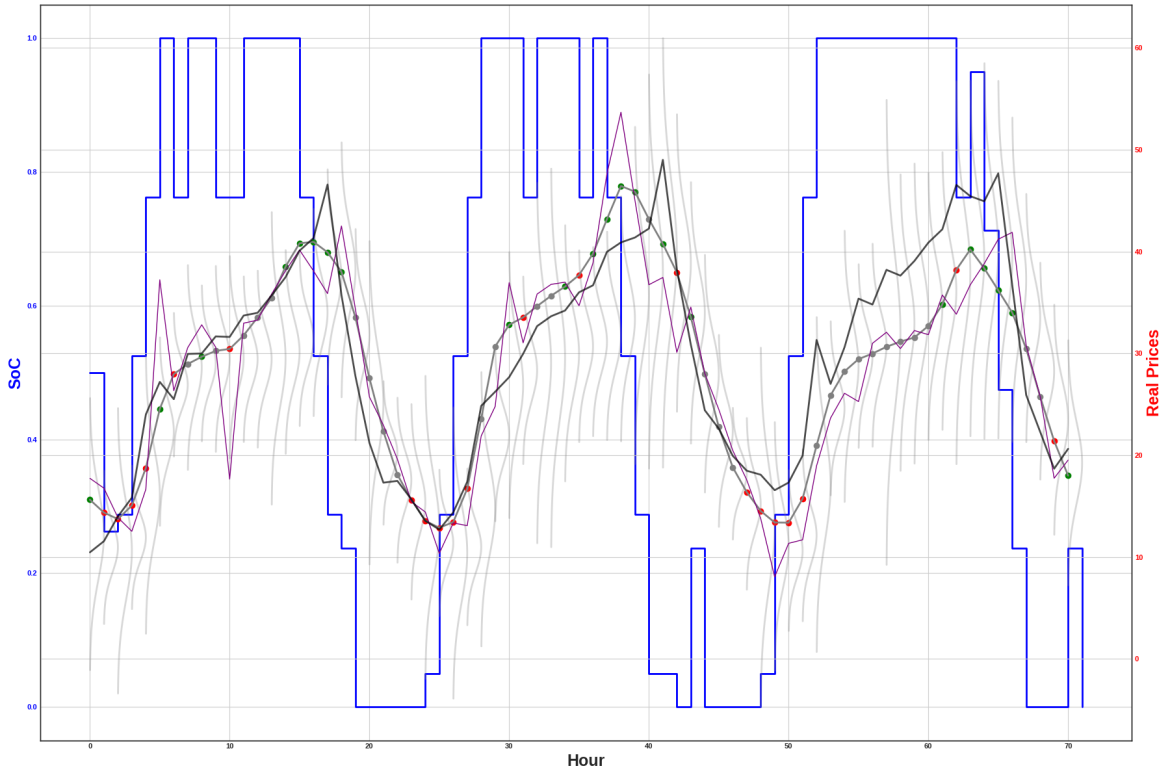


Figure 5.45: First day's strategy planned by the SMPC Optimizer.

Figures 5.50 and 5.51 show the control sequence of the BESS that the SMPC returned and the SoC along with the forecasted and actual prices over 25 days.

We also observe that the SMPC handles the limitations of charge flow of the BESS in a similar way as the MPC, as it has to spread out the charging and discharging of the battery.

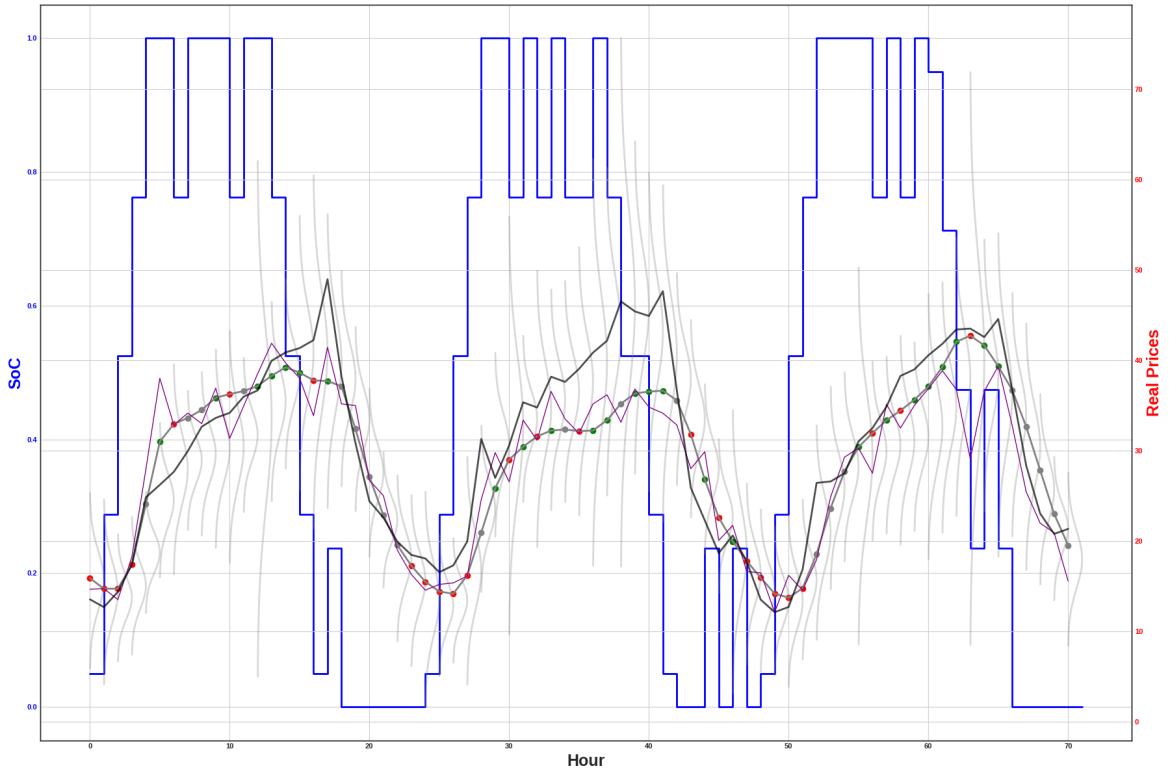


Figure 5.46: Second day's strategy planned by the SMPC Optimizer, after new knowledge occured (new prices published by NYISO).

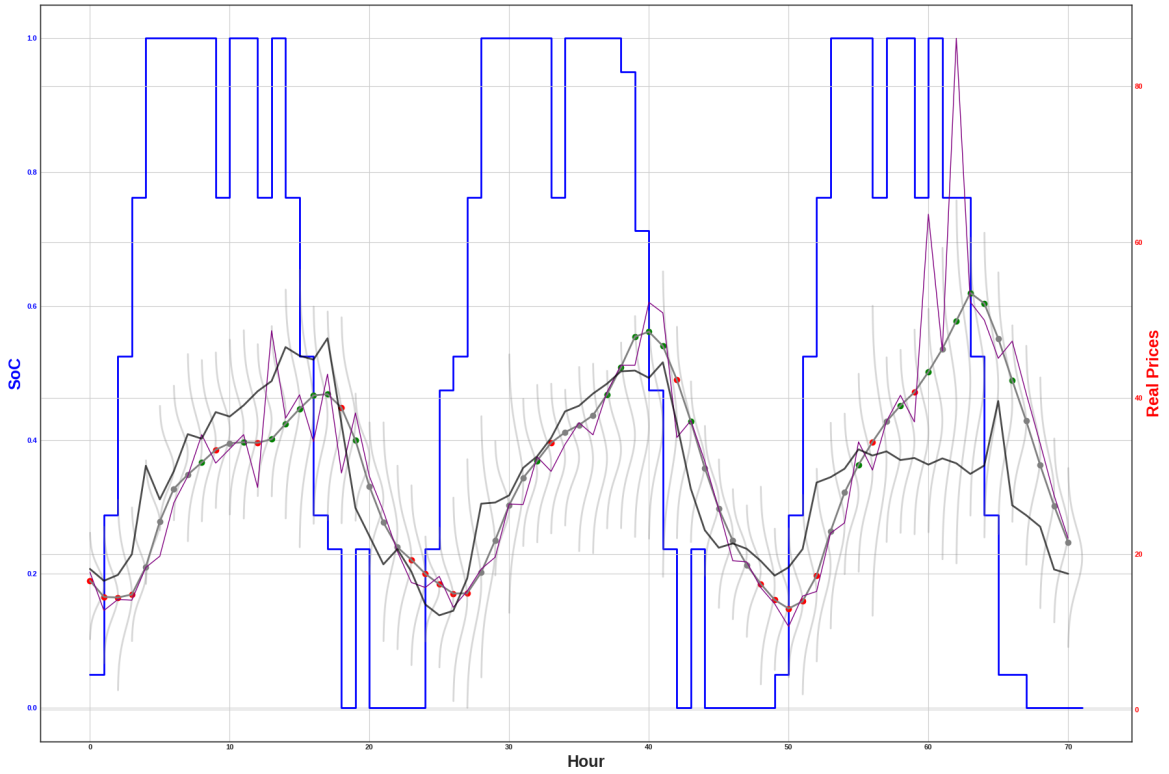


Figure 5.47: Third day's strategy planned by the SMPC Optimizer, after new knowledge occured (new prices published by NYISO).

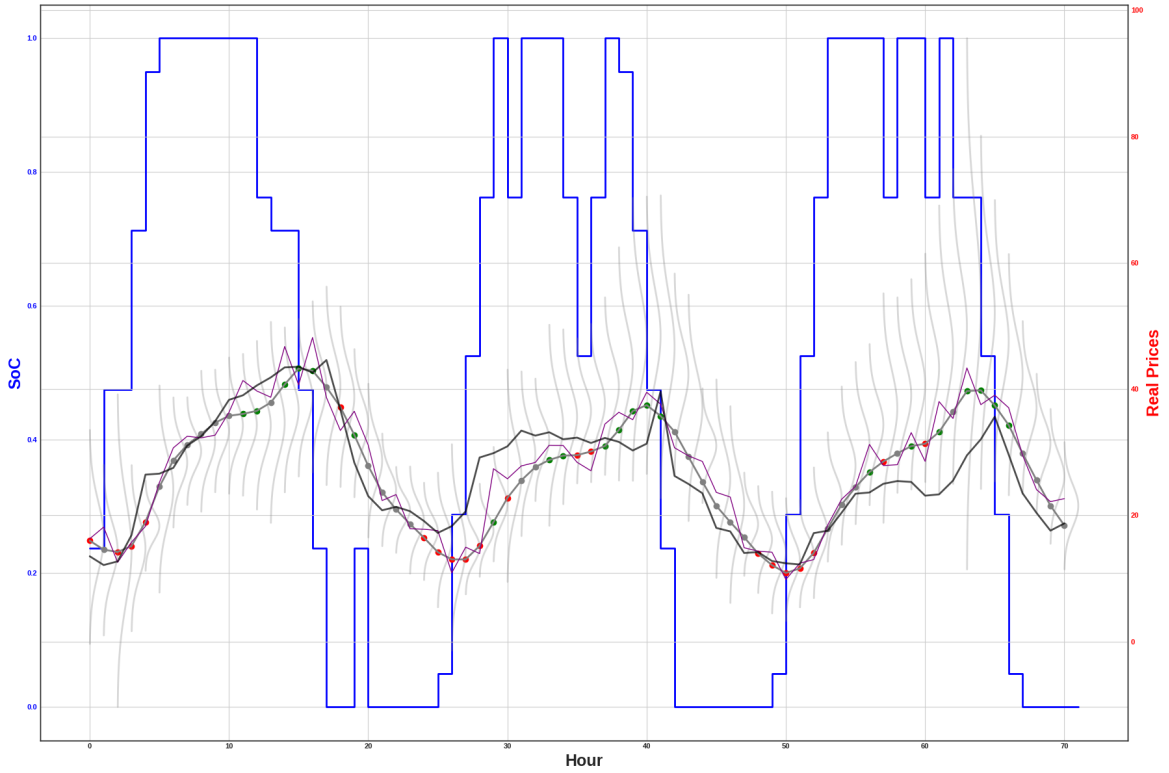


Figure 5.48: Forth day's strategy planned by the SMPC Optimizer, after new knowledge occured (new prices published by NYISO).

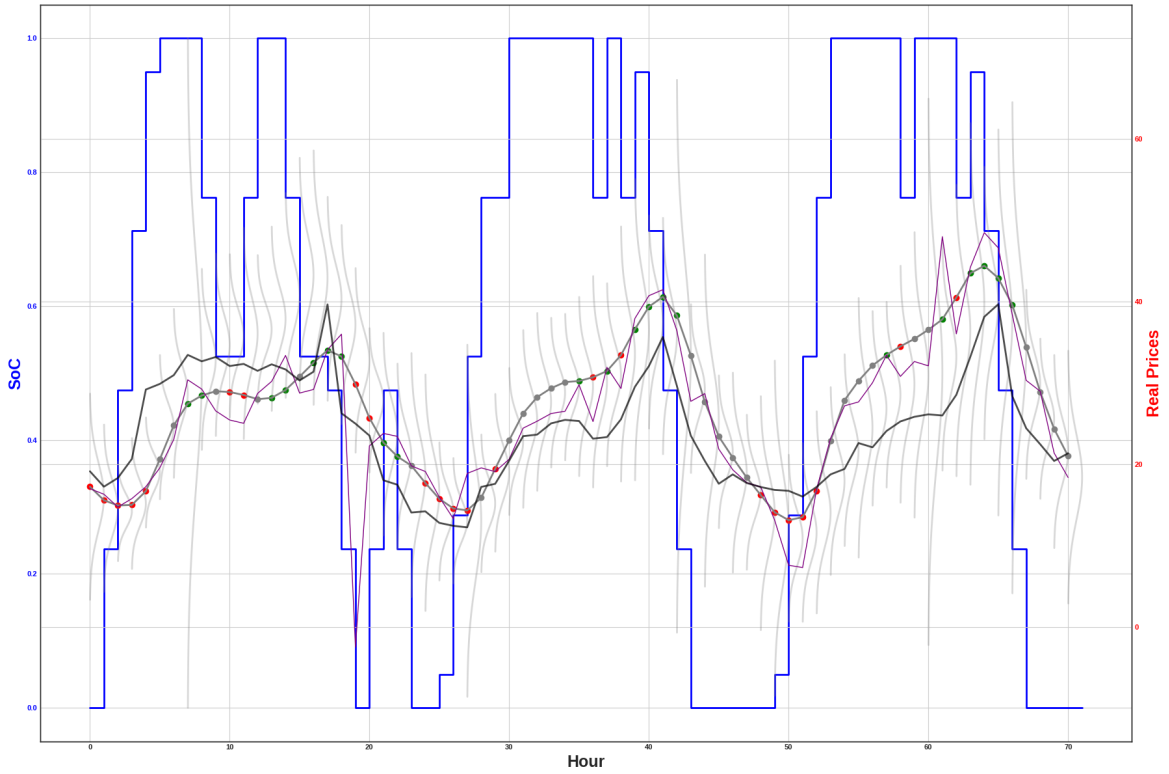


Figure 5.49: Fifth day's strategy planned by the SMPC Optimizer, after new knowledge occured (new prices published by NYISO).

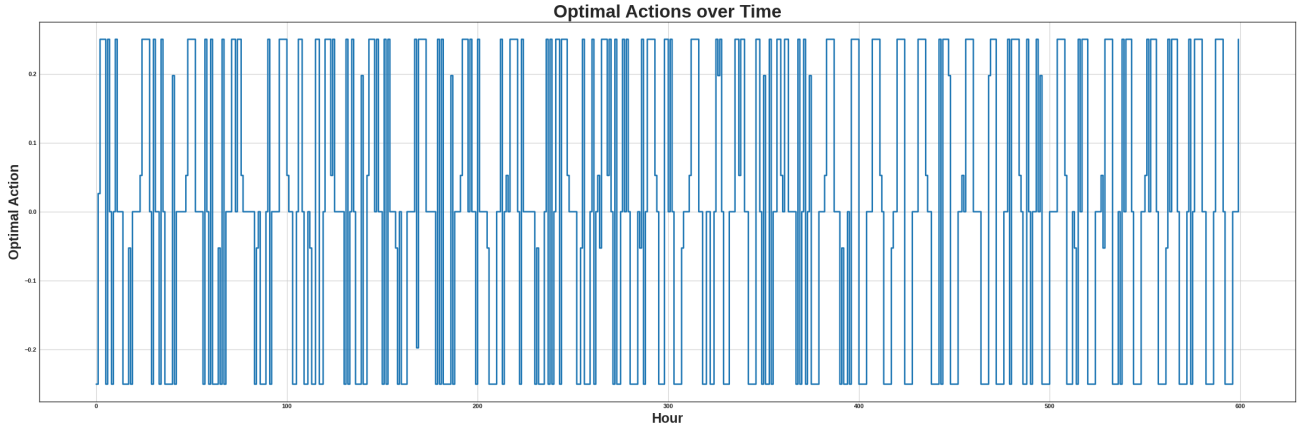


Figure 5.50: Optimal actions returned for 5 days by the SMPC optimizer.

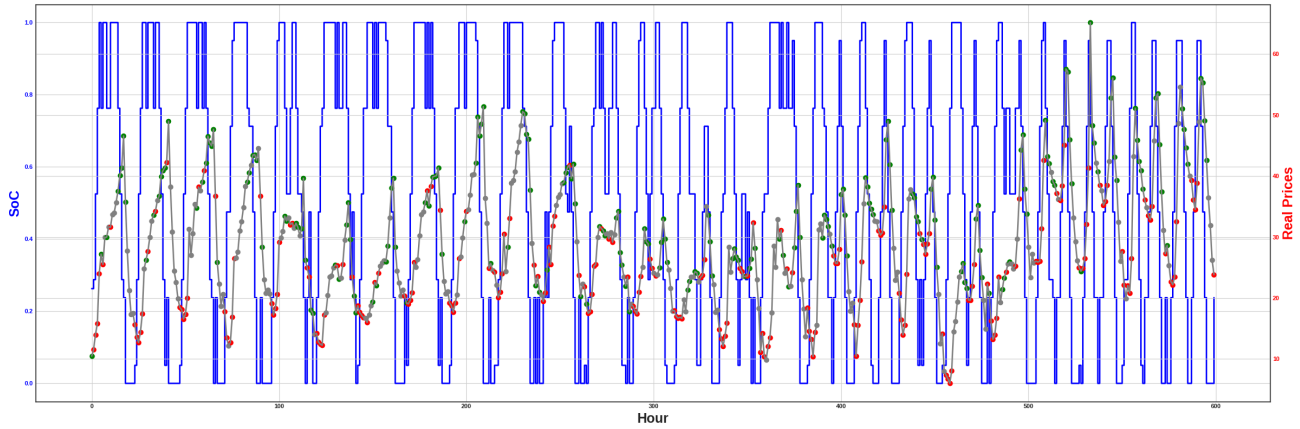


Figure 5.51: SoC levels along with fixed hourly prices for 5 days after applying the SMPC Optimizer.

5.7 Monte Carlo Simulations of Stochastic MPC in Energy Arbitrage

Monte Carlo SMPC enhances robustness by explicitly accounting for forecasting uncertainty inside the optimizer. At each decision epoch k_0, we draw M independent future-price trajectories from TimeGrad's learned distribution. Following the proposed framework for Monte Carlo SMPC, defined in Section 4.5.1, for scenario $i \in \{1, \dots, M\}$, we recursively sample

$$\hat{o}_{k+1}^{s,(i)} \sim p_\theta \left(\hat{o}_{k+1}^{s,(i)} | \mathbf{h}_k^{(i)} \right), \quad k = k_0, \dots, k_0 + N - 1,$$

where $\mathbf{h}_k^{(i)}$ is TimeGrad's internal RNN state for scenario i. Combining each stochastic sample with the known deterministic update $o_{k+1}^d = f(o_k^d, a_k)$ yields the full trajectory

$$\hat{o}_k^{(i)} = \left\langle o_k^d,\ \hat{o}_k^{s,(i)} \right\rangle \quad \text{for } k = k_0, \dots, k_0 + N.$$

Each scenario's cumulative reward is

$$\mathcal{J}^{(i)} = \sum_{k=k_0}^{k_0+N-1} R\big(\hat{o}_k^{(i)},\, a_k\big).$$

The SMPC then selects the control sequence that maximizes the average performance across all M trajectories:

$$\begin{aligned} \underset{a_{k_0}, \dots, a_{N+k_0-1}}{\text{maximize}} \quad & \frac{1}{M} \sum_{i=1}^{M} \mathcal{J}^{(i)} \\ \text{subject to} \quad & o_0, \dots, o_{k_0-1} \text{ observed}, \\ & \hat{o}_{k+1}^d = f(\hat{o}_k^d, a_k),\ \hat{o}_{k_0} = o_{k_0} \\ & \hat{o}_{k+1}^{s,(i)} \sim p_\theta \left(\hat{o}_{k+1}^{s,(i)} | \mathbf{h}_k^{(i)} \right), \\ & a_k \in \mathcal{A}, \quad k = k_0, \dots, N-1. \end{aligned} \tag{5.1}$$

By optimizing the expected reward over multiple realizations, Monte Carlo SMPC plans control actions that are robust to the complete distribution of future prices, balancing aggressive and conservative actions, and also mitigates the problem of the disturbed forecasts, since the noise is averaged out. Although Monte Carlo simulations are computationally expensive, they offer a powerful way to manage uncertainty. Like the previous controllers, the control strategy adapts to the stochastic market conditions. Figures 5.52 to 5.56 show the control strategies that the Monte Carlo SMPC plans over five consecutive days in the market by considering 100 realizations of the future system states. Once the optimal strategy is calculated, the agent performs the best actions for the first day's prices and moves to the next day, when new data is available.

Figures 5.57 and 5.58 show the control sequence of the BESS that the Monte Carlo SMPC returned and the SoC along with the prices of the specific region over 25 days. Specifically, for the Figure 5.58, the bold blue line refers to the SoC of the BESS for each time step, the gray dotted line is the region's prices for each time step. The color of the dots on the gray line help understand the action, following the same logic as in MPC and SMPC. The thin purple lines represent the realizations of the stochastic model of the system.

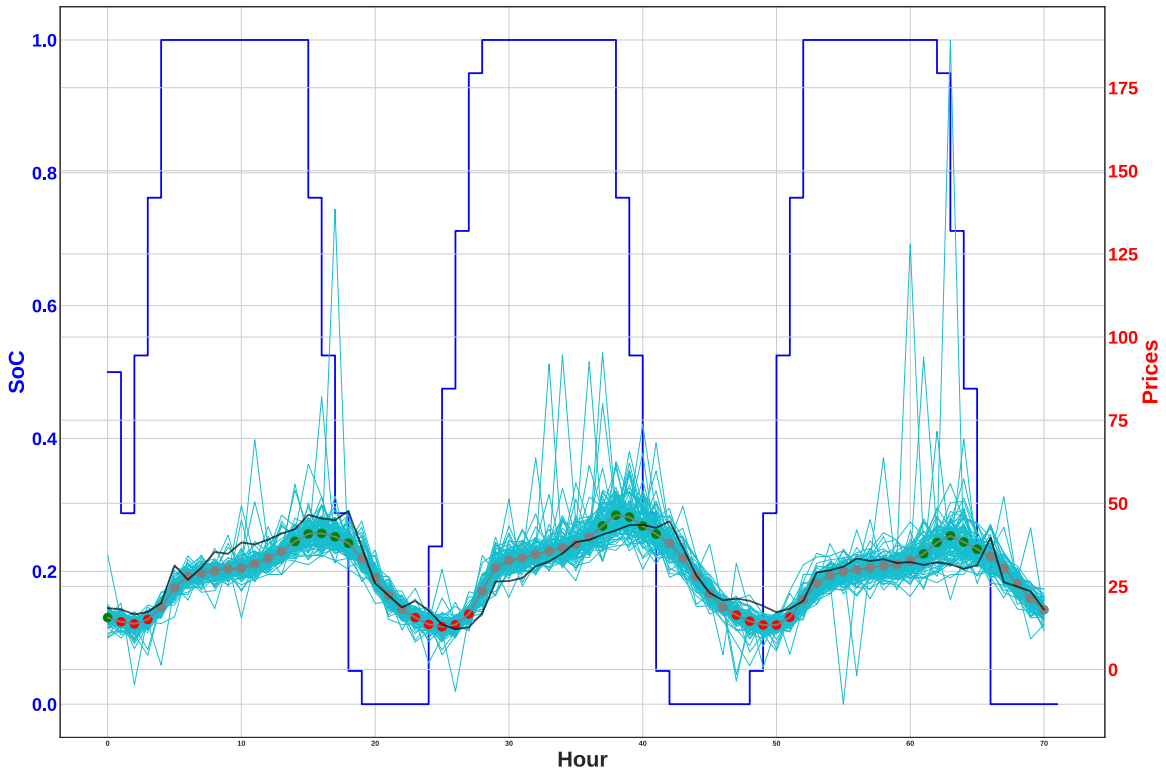


Figure 5.52: First day's strategy planned by the Monte Carlo SMPC Optimizer.

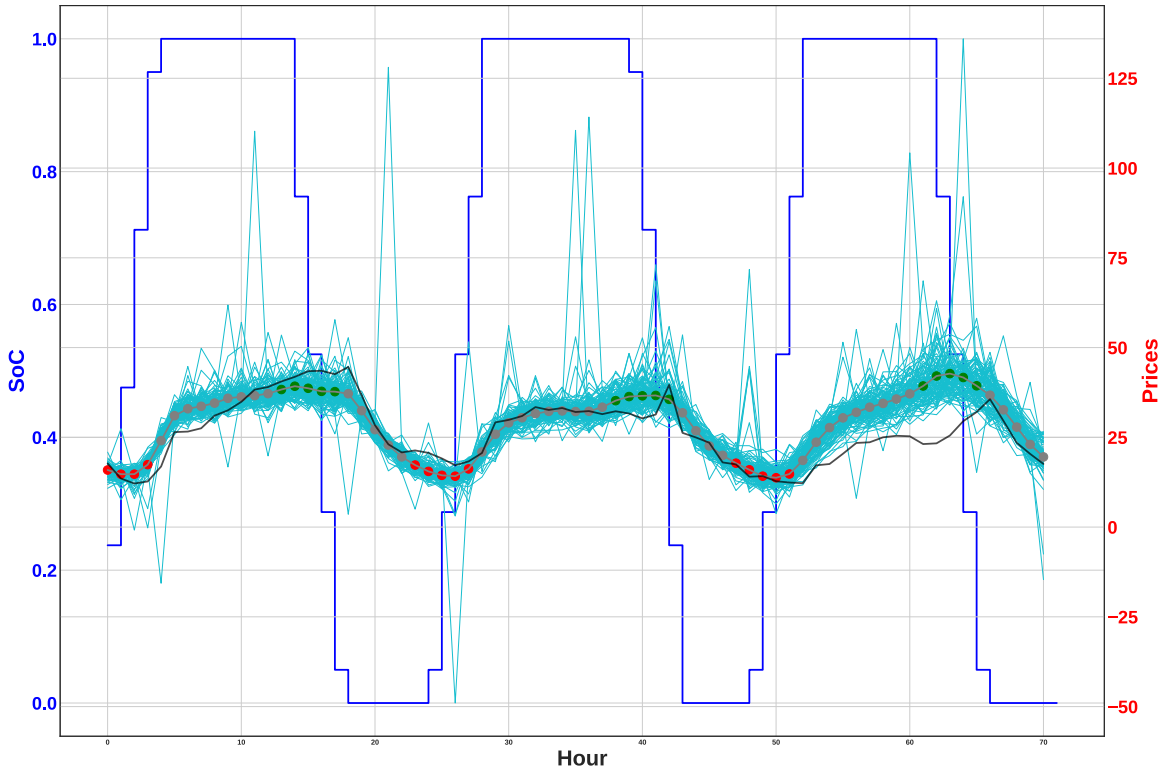


Figure 5.53: Second day's strategy planned by the Monte Carlo SMPC Optimizer, after new knowledge occured (new prices published by NYISO).

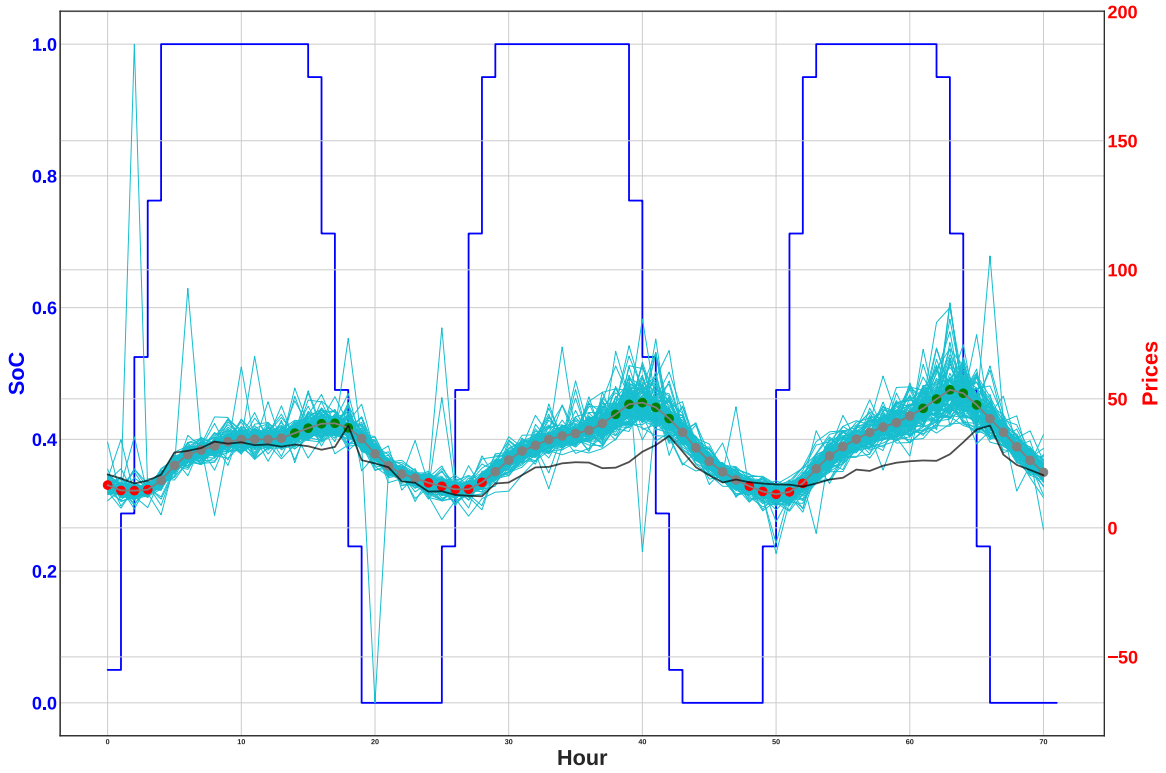


Figure 5.54: Third day's strategy planned by the Monte Carlo SMPC Optimizer, after new knowledge occured (new prices published by NYISO).

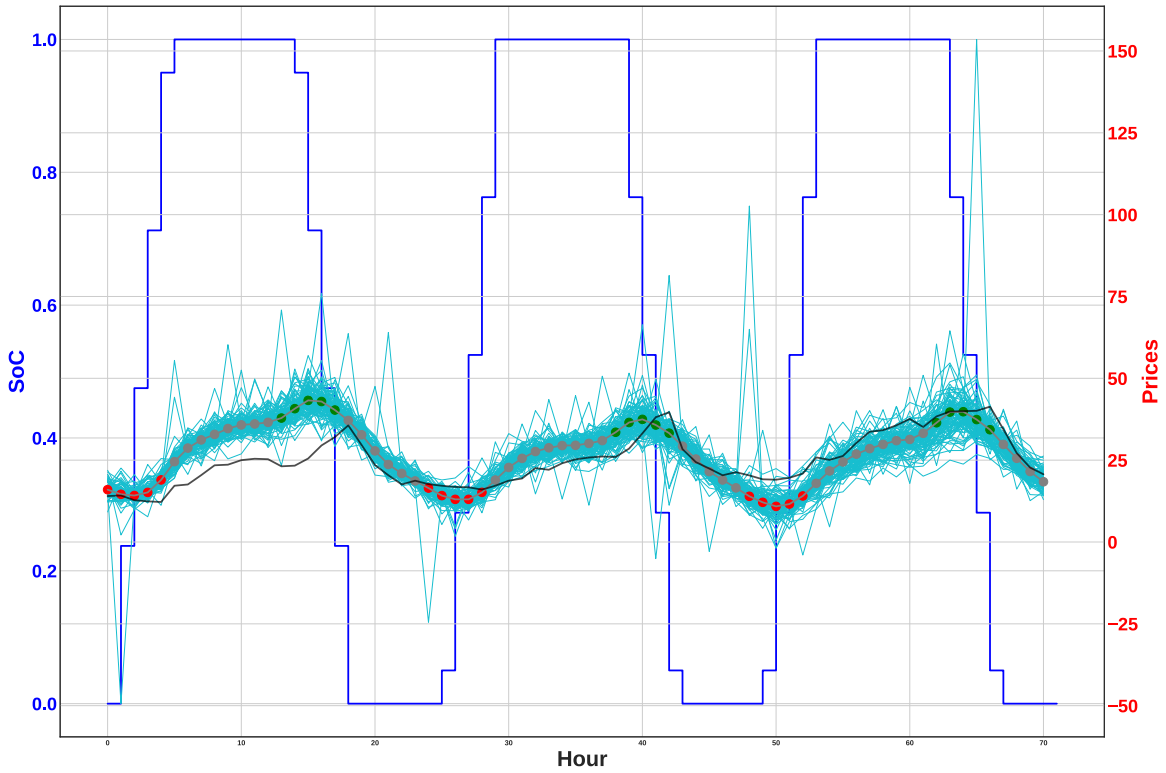


Figure 5.55: Forth day's strategy planned by the Monte Carlo SMPC Optimizer, after new knowledge occured (new prices published by NYISO).

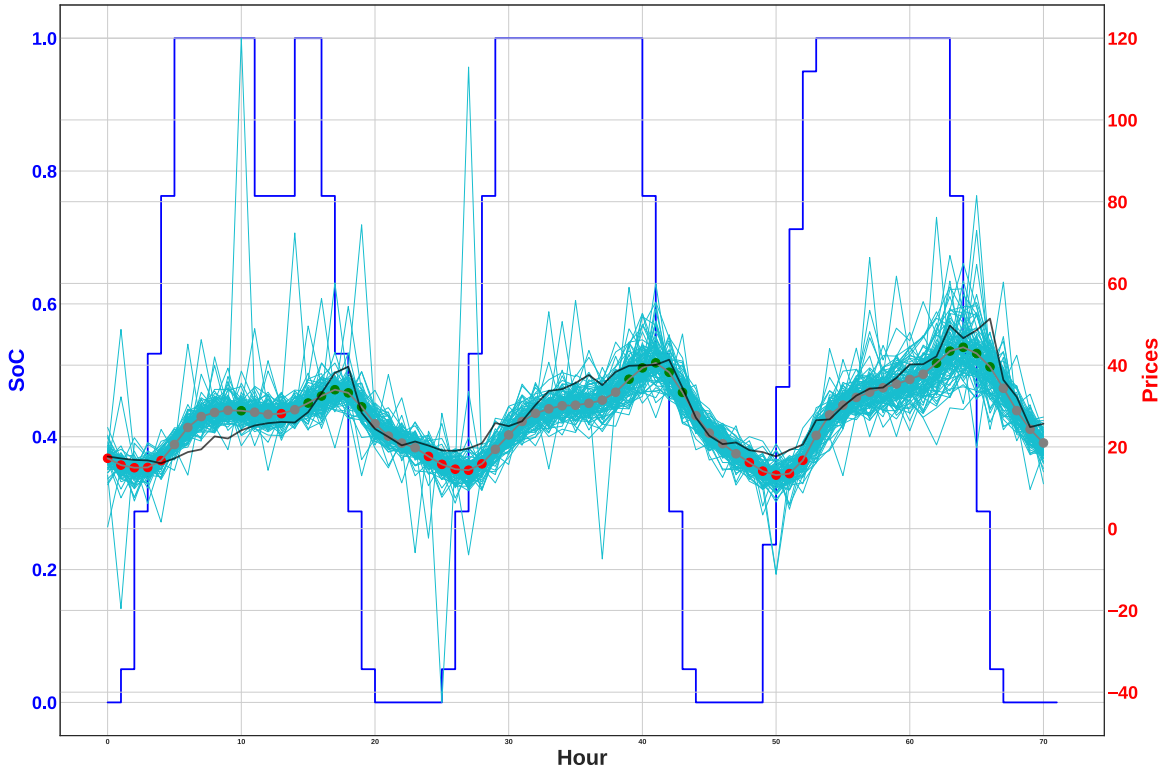


Figure 5.56: Fifth day's strategy planned by the Monte Carlo SMPC Optimizer, after new knowledge occured (new prices published by NYISO).

The small gray gaussians for every predicted time step show the distributions of which the trajectories' points are sampled.

By looking at the dots' colors and the thin purple lines in the Figures 5.52 to 5.56 we can see that the BESS now operates better than in the case of SMPC. Again, in general, it buys energy when prices are low and sells when prices are high and it charges the BESS when it anticipates prices to increase and discharges when prices peak, but also the variability of each individual trajectory does not really affect the planning. On the contrary, the multiple realizations of the system's stochastic dynamics are a robust approach in planning under the uncertainty of the system dynamics.

Once again, the Monte Carlo SMPC can easily handle the limitations of charge flow of the BESS in a similar way as the MPC and SMPC.

However, the limitation of the finite horizon optimization remains unsolved. We seek the solution in the section where we examine Multistep Lookahead Rollout algorithms.

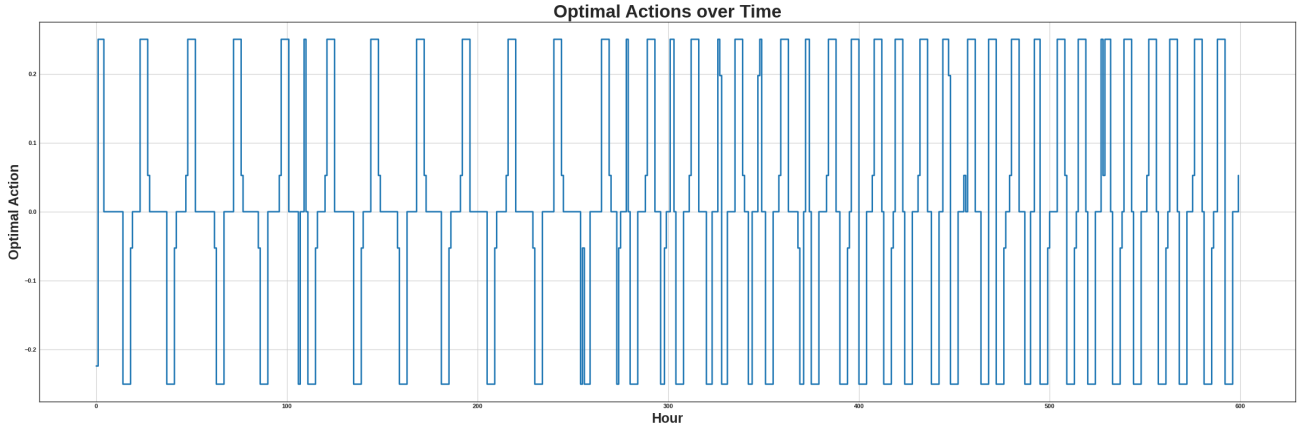


Figure 5.57: Optimal actions returned for 5 days by the Monte Carlo SMPC optimizer.

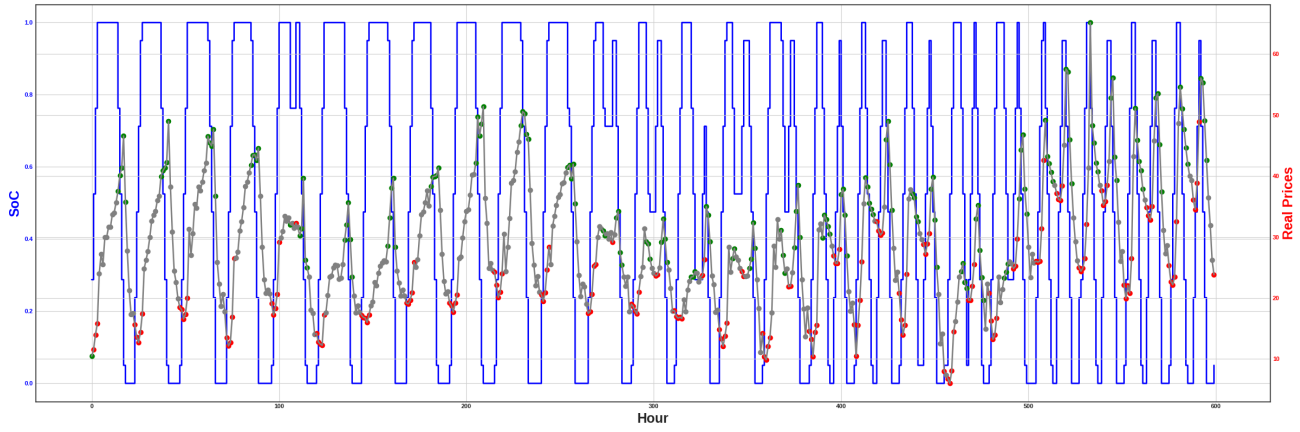


Figure 5.58: SoC levels along with fixed hourly prices for 5 days after applying the Monte Carlo SMPC Optimizer.

5.8 Scenario Tree–Based MPC in Energy Arbitrage

Scenario Tree-based Model Predictive Control offers a structured approach to managing uncertainty in energy arbitrage by discretizing future uncertainties into a scenario tree. This method enables the optimization of control actions over multiple possible future evolutions of uncertain variables, which in our case are the energy prices.

At each decision epoch k_0, a scenario tree $\mathcal{T}$ of depth $N = 3$ is built, as described in Section 4.5.2, whose nodes $i \in \mathcal{T}_t$ at stage $t = k_0, \ldots, k_0 + N$ carry:

- The forecasted observation $\hat{o}_t^i = \langle \hat{o}_t^{i,d},\ \hat{o}_t^{i,s} \rangle$,
- The branching probability π_i,
- A pointer to the parent node $\mathrm{pre}(i)$.

We then solve the following stochastic MPC problem over $\mathcal{T}$, choosing one action a_t^i per node:

$$
\begin{aligned}
\underset{\{a_t^i\}}{\text{maximize}} \quad & \sum_{k=k_0}^{k_0+N-1} \sum_{i \in \mathcal{T}_k} \pi_i \; R\big(\hat{o}_t^i,\ a_t^i\big) \\
\text{subject to} \quad & \hat{o}_{k_0}^1 = o_{k_0}, \\
& \hat{o}_{k+1}^{i,d} \;=\; f\big(\hat{o}_k^{\mathrm{pre}(i),d},\ a_k^{\mathrm{pre}(i)}\big) \quad \forall\, i \in \mathcal{T}_{k+1} \setminus \{1\}, \\
& a_k^i = a_k^j \quad \text{whenever nodes } i, j \text{ share the same history up to } k, \\
& a_k^i \in \mathcal{A} \quad \forall\, i, k.
\end{aligned}
$$

Here:

- $\mathcal{T}_k \subset \mathcal{T}$ is the set of nodes at stage k,
- node 1 is the root at $k = k_0$,
- π_i is the product of branch-probabilities along the path from the root to i, i.e. the realization of the stochastic variables up to time i,
- $f(\cdot)$ is the known deterministic evolution of the SoC,
- non-anticipativity $a_t^i = a_t^j$ enforces that decisions depend only on the shared history.

This formulation maximizes the probability-weighted sum of immediate rewards R over all tree nodes, subject to the battery dynamics and non-anticipativity. In essence, this approach determines the optimal sequence of battery actions across multiple potential future scenarios, recognizing that each action's outcome influences the future decisions and system trajectory. We implement the two versions of generating the stochastic system's scenario tree proposed in Section 4.5.2.

Intuition Behind the Scenario tree: The tree diagram in Figure 5.59 visualizes the concept of how the Scenario Tree captures the uncertainty of the stochastic process. It illustrates how the continuous probability distribution of future stochastic variables (energy prices) is discretized into a finite set of distinct scenarios. At each stage, the tree branches out, with each branch signifying a possible *realization* of the stochastic system's evolution. Each node, carries a forecasted observation and the arrows connecting these nodes represent transitions between stages, and contain the probability $P(\cdot)$ of that specific branch occurring and the control action $a_{(\cdot)}$ taken at that stage, which is the decision made at the preceding node to navigate towards that potential future.

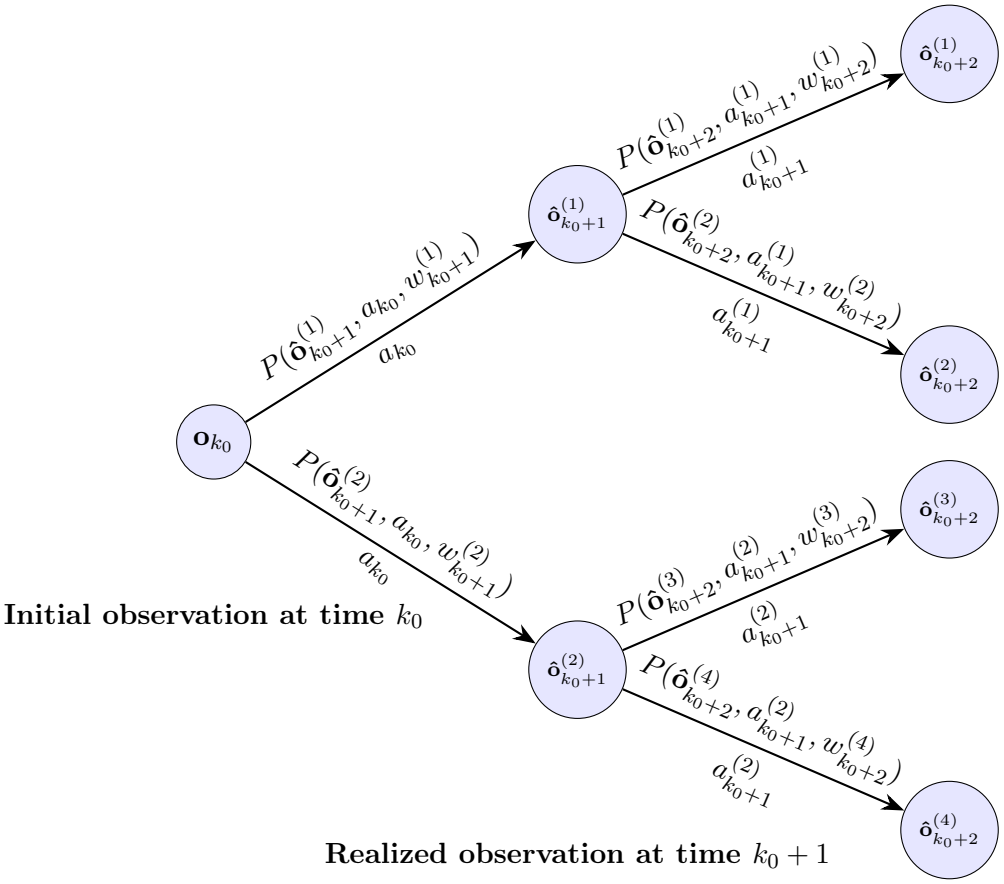


Figure 5.59: Visualization of a Scenario Tree that discretizes the probability distribution of the future steps.

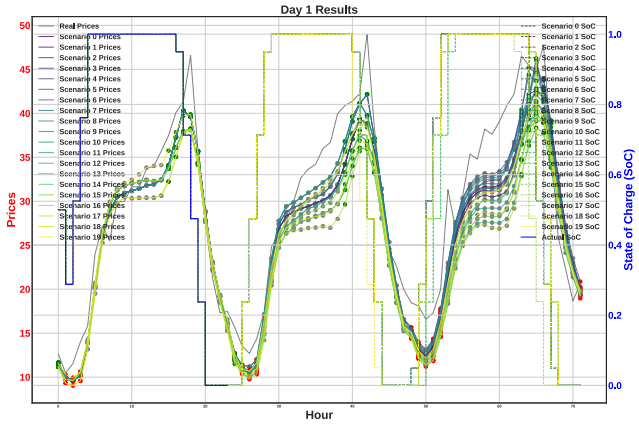
Receding Horizon: The Scenario Tree-Based MPC approach also uses the receding horizon concept, where the optimization is performed over a finite horizon, and after deciding the control actions for the first stage (day), the horizon is shifted 1 stage (24 time steps) forward, and the whole Scenario Tree is generated again, in order to optimize on the updated data and forecasts.

5.8.1 Forward-Clustering Scenario Tree MPC

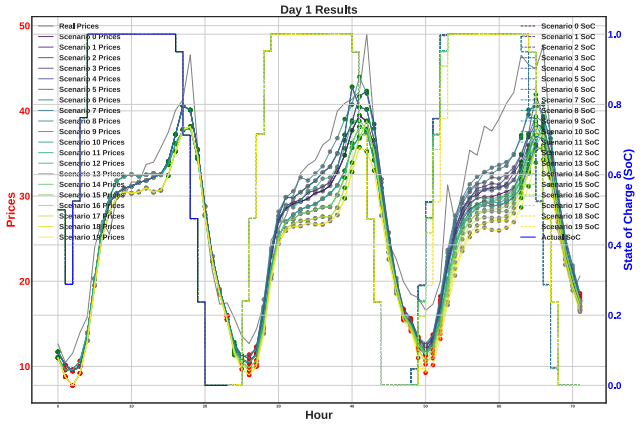
To apply the forward-clustering variant of Scenario Tree–Based MPC to the energy arbitrage problem, we grow a tree over a planning horizon of $N = 3 \cdot 24$ steps (3-day horizon), in $D = 3$ stages, by iteratively sampling and clustering TimeGrad forecasts. At staging step k_0, given the history of observed stochastic prices $o_0^s, \ldots, o_{k_0-1}^s$, we draw $M = 5000$ sample forecasts $\{\mathcal{F}^{(i)}\}_{i=1}^{M}$ of length $H = 24$ (the length of one full stage, to create the branches) from TimeGrad, cluster them via K-means into K groups, and take each cluster centroid μ_k as a representative forecast for the correspondent node with probability p_k. For this version, we examined how the algorithm performs by taking the mean and the median as a centroids in two different implementations. We then trim to the top $L = 20$ branches by probability (to avoid the exponential expansion) and repeat this at each node until depth $D = 3$. The resulting tree has nodes $n \in \mathcal{N}_k$ at stage k, each with forecasted observation $\hat{o}_k^n$ and cumulative path probability π_n.

Then the multistage MPC optimization is solved over this tree to maximize the expected profit, taking into consideration all the possible generated scenarios.

The Forward-Clustering Scenario Tree-Based MPC provides a robust control strategy by explicitly considering multiple future scenarios. Figures 5.60 to 5.64 show the control strategies planned over five consecutive days in the market. The left plots refer to the version where the median is used as the cluster centroid, while in the right plots, the mean is the centroid.



Scenario Tree-Based MPC using medians as centroids.

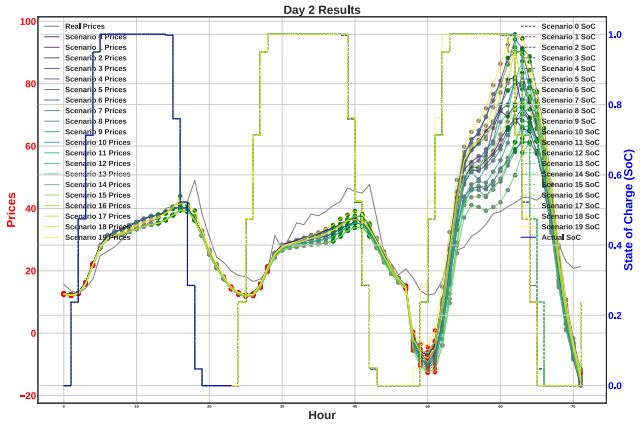


Scenario Tree-Based MPC using means as centroids.

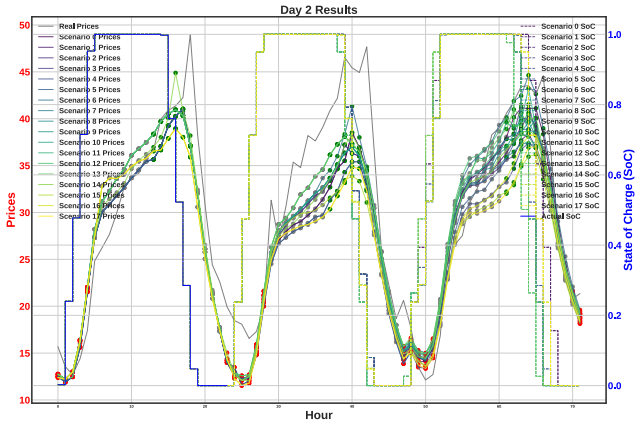
Figure 5.60: First day's strategy planned by the Forward-Clustering Scenario Tree-Based MPC Optimizer for two versions.

Figures 5.60 to 5.64 illustrate the daily control strategies generated by the Forward-Clustering Scenario Tree MPC. Both the left (median as centroid) and right (mean as centroid) columns depict the planned actions over five consecutive days.

Each graph visualizes the scenario tree constructed for three days (stages) ahead. In each stage the scenarios branch out, with every 24-hour forecasted price prediction corresponding to each node of the stage. For each potential future scenario, the agent's actions have colors as the previous implementations of MPC: red dots indicate moments when the agent decides to buy energy, green dots refer to the agent selling amounts of energy, and grey dots show time steps of no action. A distinct grey trajectory on each plot represents the actual, realized energy prices, in order to visualize the accuracy of the predictor.

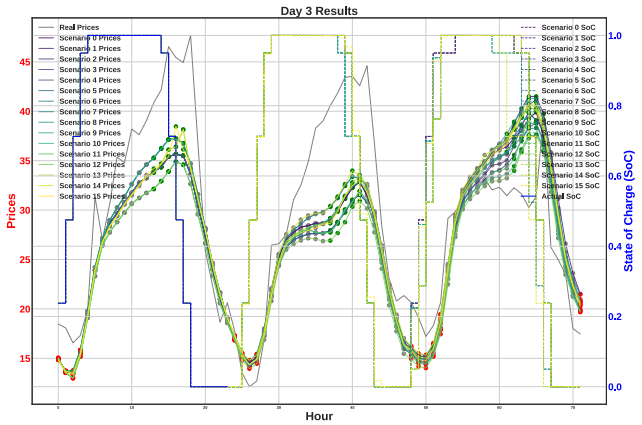


Scenario Tree-Based MPC using medians as centroids.

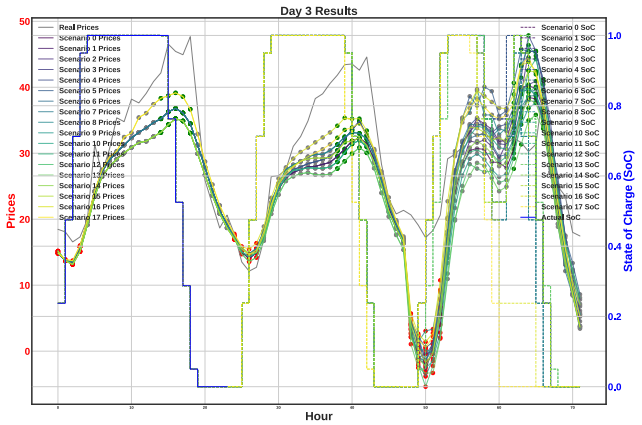


Scenario Tree-Based MPC using means as centroids.

Figure 5.61: Second day's strategy planned by the Forward-Clustering Scenario Tree-Based MPC Optimizer for two versions.

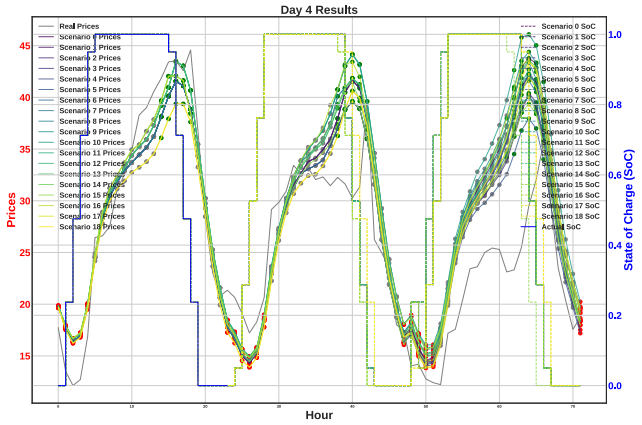


Scenario Tree-Based MPC using medians as centroids.

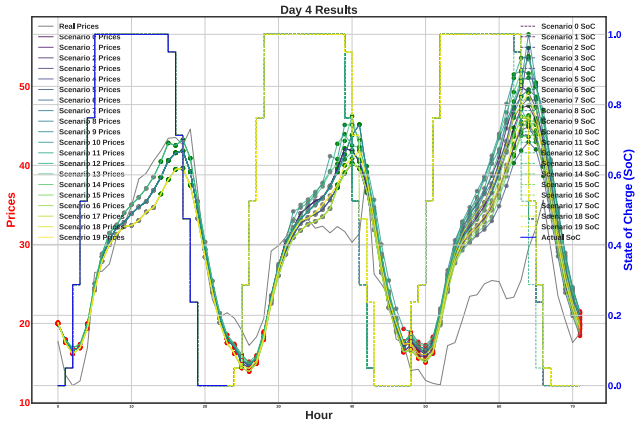


Scenario Tree-Based MPC using means as centroids.

Figure 5.62: Third day's strategy planned by the Forward-Clustering Scenario Tree-Based MPC Optimizer for two versions.

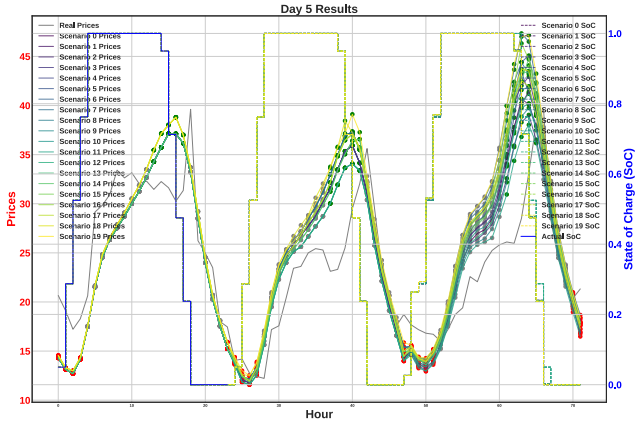


Scenario Tree-Based MPC using medians as centroids.

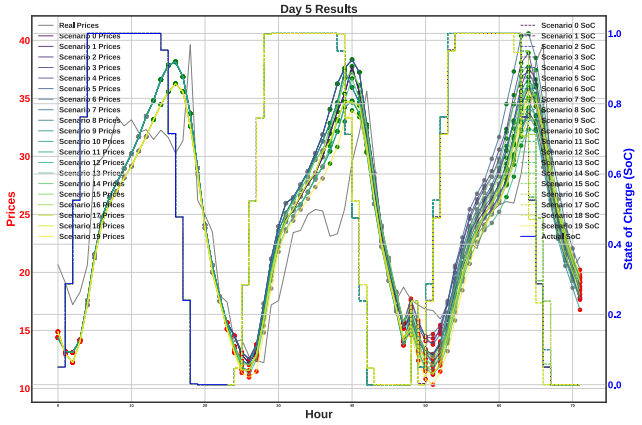


Scenario Tree-Based MPC using means as centroids.

Figure 5.63: Fourth day's strategy planned by the Forward-Clustering Scenario Tree-Based MPC Optimizer for two versions.



Scenario Tree-Based MPC using medians as centroids.



Scenario Tree-Based MPC using means as centroids.

Figure 5.64: Fifth day's strategy planned by the Forward-Clustering Scenario Tree-Based MPC Optimizer for two versions.

The control actions are initially identical at the root of the tree (the current decision epoch), which are the ones to be applied after the optimization. As the tree branches out, the actions diverge, reflecting the agent's adaptive plan based on the realization of the stochastic variables (energy prices) along each path. One can observe the non-anticipative nature of the Forward-Clustering Scenario Tree MPC algorithm, where future decisions are evaluated as new information unfolds (stochastic variables are realized).

Overall, the agent exhibits highly effective behavior in the energy market. It consistently manages to buy energy at low prices and sell at high prices, demonstrating a robust ability to use market fluctuations to maximize the long-term goal. Furthermore, the multi-stage optimization inherent in the scenario tree approach enables the agent to plan for long-term profits by strategically considering a multitude of possible future scenarios, leading to an adaptive and profitable control strategy.

5.8.2 Backward-Hierarchical Scenario Tree MPC

In the backward-hierarchical variant, we first sample a full set of $M = 5000$ price trajectories $\{x^{(i)}\}_{i=1}^{M}$ over the entire horizon $N = 3 \cdot 24$ from TimeGrad's generated distribution. For the $D = 3$ stages of the optimization, beginning at the final stage $k = k_0 + D$, we cluster these $M = 5000$ samples into $K_D = 20$ groups to form leaf-nodes (ultimately resulting in $K_D = 20$ scenarios). We then merge each group's trajectories and recluster into K_{D-1} parent nodes, and so on, up to the root, ensuring clusters only coalesce and thus enforcing non-anticipativity by construction. The optimal clustering number $K_d, d < D$ is found by iteratively searching for it, keeping the one that optimizes the clustering objective function (as described in our backward clustering tree generation algorithm). Each node n at stage k has a centroid μ_k^n and probability p_k^n equal to the fraction of samples in that cluster.

Then the same multistage MPC optimization is solved over this tree version to maximize the expected profit over all the scenarios. Figures 5.60 to 5.64 show the control strategies planned over five consecutive days in the market. The left plots refer to the version where the median is used as the cluster centroid, while in the right plots, the mean is the centroid.

The strategies generated by the Backward-Hierarchical Scenario Tree MPC are presented in Figures 5.65 to 5.69. Similar to the forward-clustering approach, each plot showcases the agent's actions over a day, visualized within the constructed scenario tree.

Again, each node within this tree also represents a 24-hour forecasted price prediction, and the branching illustrates the different possible price evolutions. The agent's decisions are again indicated by red dots for buying, green dots for selling, and grey dots for no action. The grey trajectory on each plot represents the actual, realized energy prices.

The plots demonstrate that the control actions are the same at the root but progressively diverge as the tree branches out, respecting the non-anticipativity constraints. Like in the forward-clustering variant, this Backward-Hierarchical approach also allows the agent to effectively buy energy at low prices and sell at high prices, thereby optimizing energy arbitrage.

Ultimately, when comparing the strategies derived from both the Forward-Clustering and Backward-Hierarchical Scenario Tree MPC methods, one can see that the results are similar, with both approaches yielding effective control strategies. This will be viewed more formally later, where the metrics of the various MPC algorithms are compared.

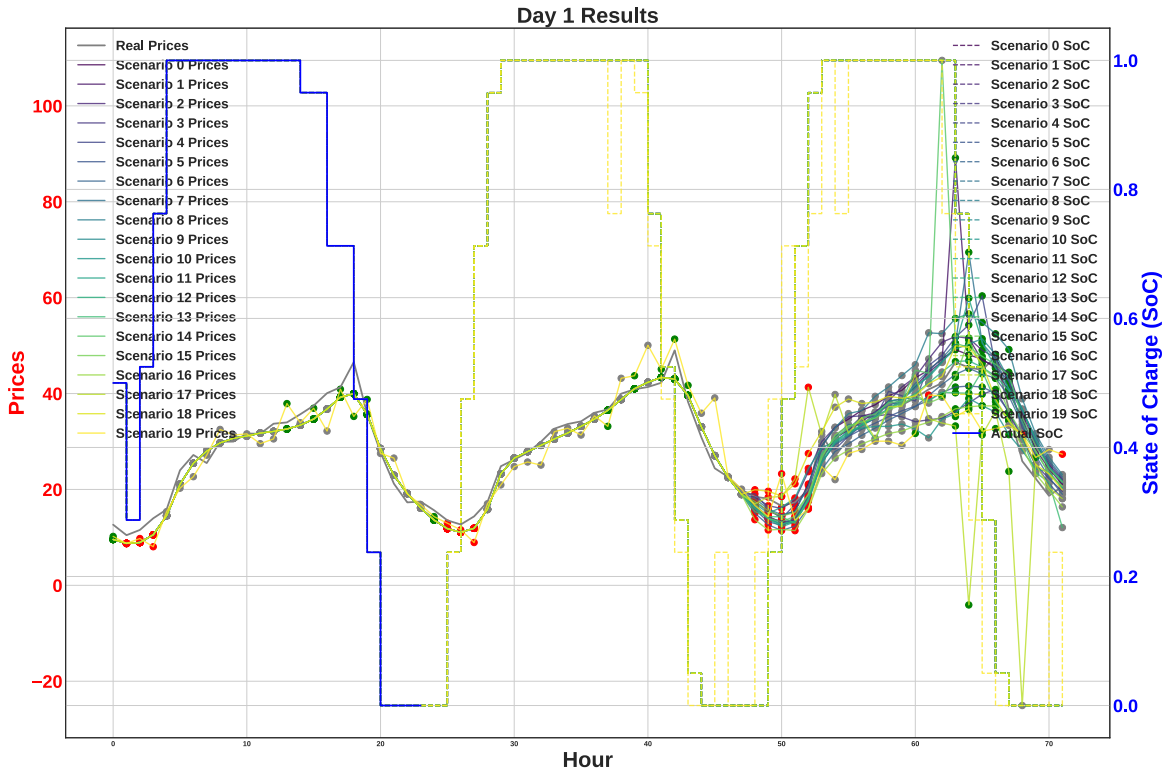


Figure 5.65: First day's strategy planned by the Backward-Hierarchical Scenario Tree-Based MPC Optimizer.

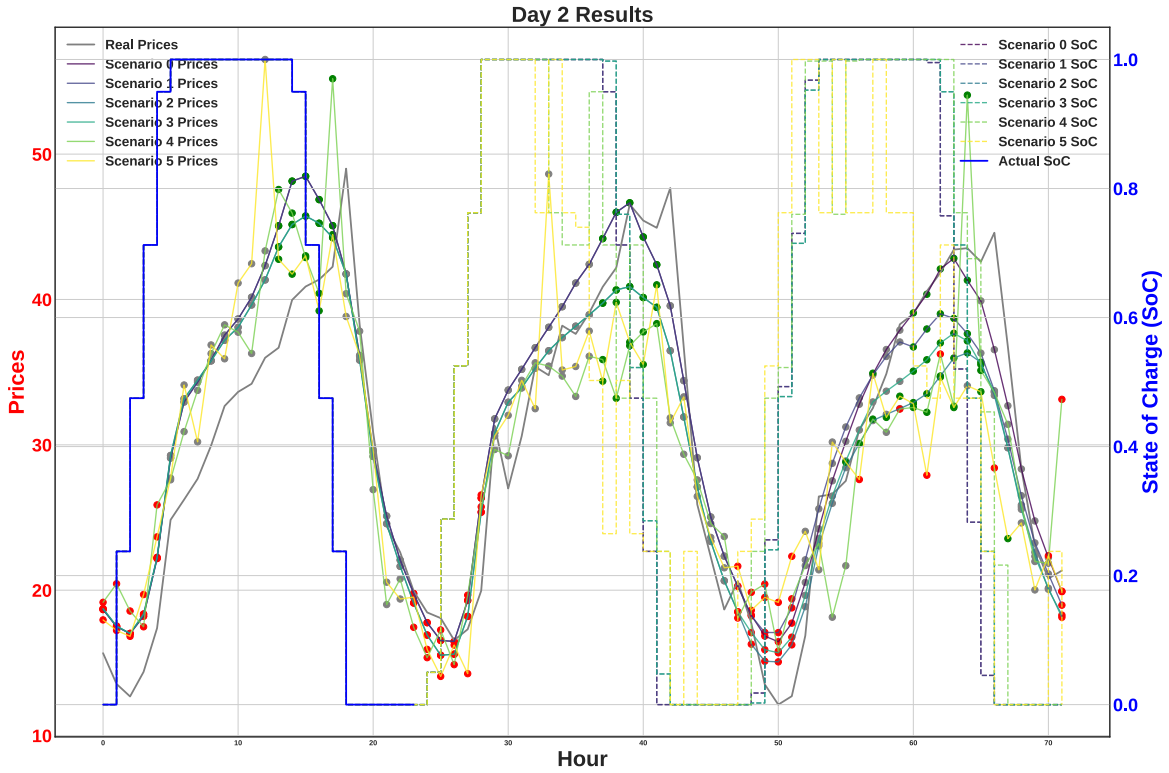


Figure 5.66: Second day's strategy planned by the Backward-Hierarchical Scenario Tree-Based MPC Optimizer.

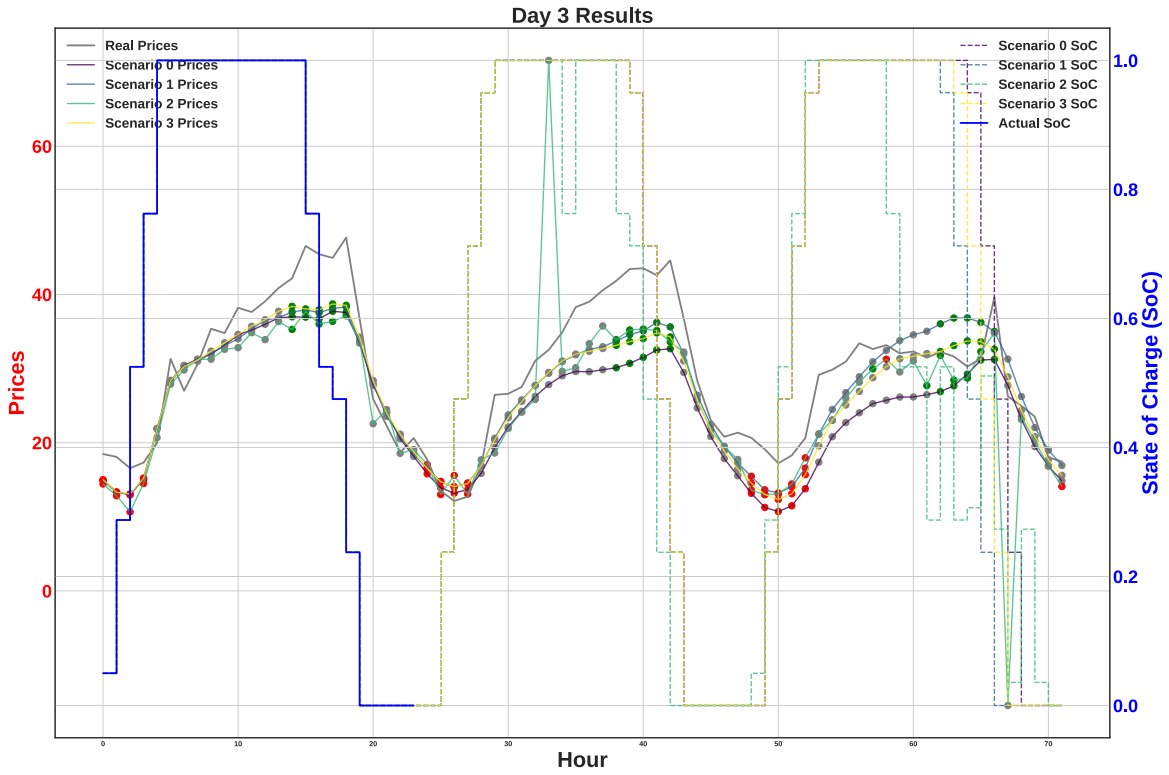


Figure 5.67: Third day's strategy planned by the Backward-Hierarchical Scenario Tree-Based MPC Optimizer.

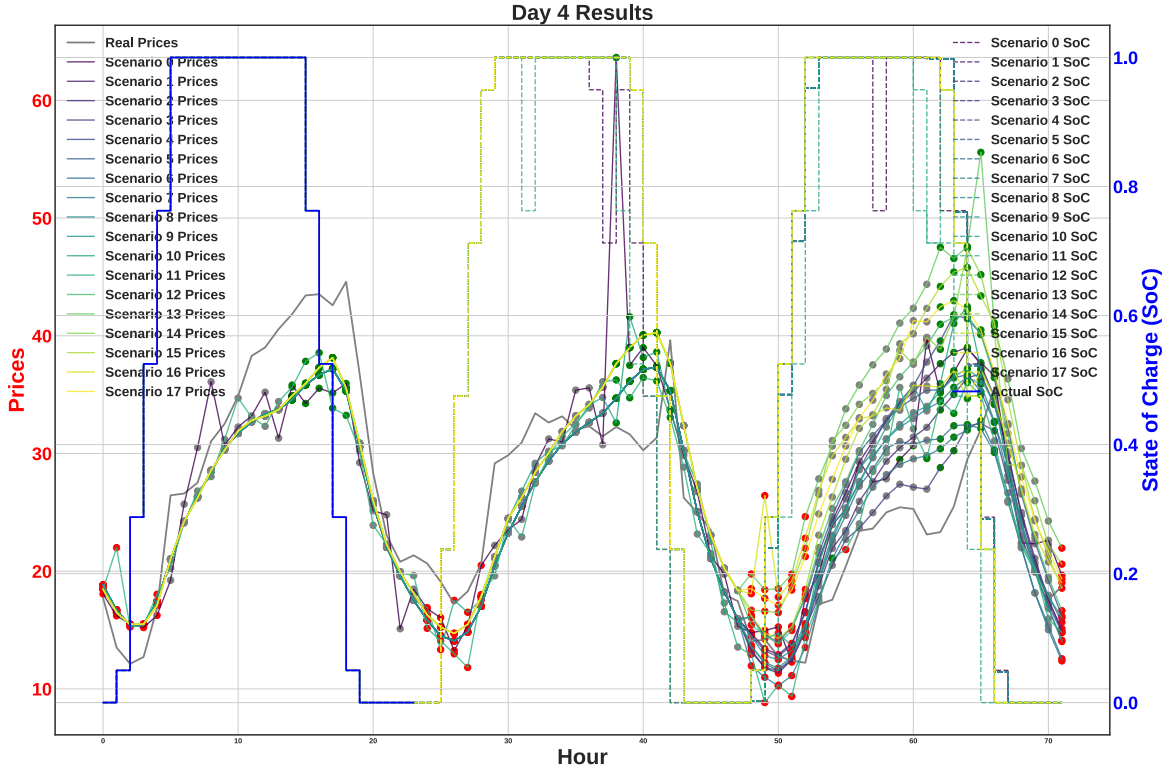


Figure 5.68: Forth day's strategy planned by the Backward-Hierarchical Scenario Tree-Based MPC Optimizer.

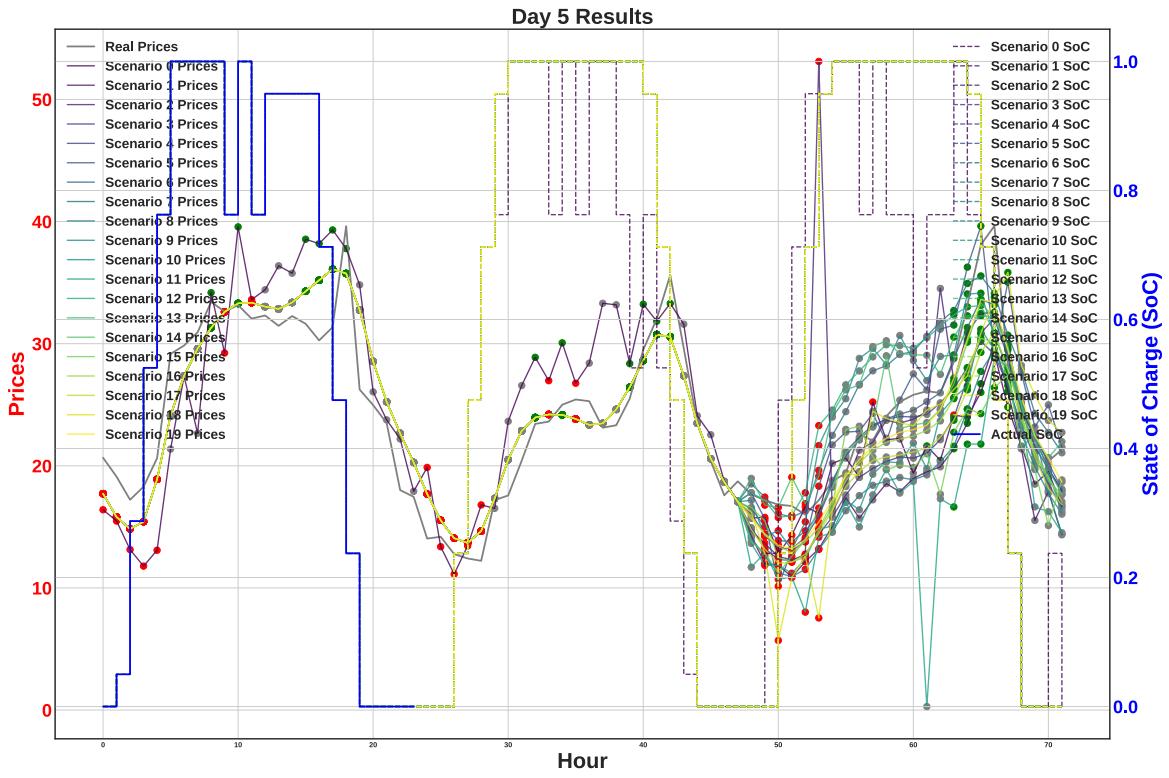


Figure 5.69: Fifth day's strategy planned by the Backward-Hierarchical Scenario Tree-Based MPC Optimizer.

5.9 Heuristic-Augmented MPC in Energy Arbitrage

Heuristic-Augmented MPC (H-A MPC) extends the Multistep Lookahead techniques we have examined so far, by incorporating a heuristic function at the end of the horizon. This usually is a function of the terminal state or observation that estimates the expected reward that the agent will achieve after the lookahead horizon, if it ends up in the given terminal state. An illustration of the extension is depicted in Figure 5.70, where the agent examines both the possible actions in the prediction horizon and the estimated gain of ending in various states. Following the framework defined in Section 4.6, instead of a function of the terminal observation, we create a function of the observations that estimates the optimal terminal observation that the system has to pass through to maximize the expected reward after the end of the horizon.

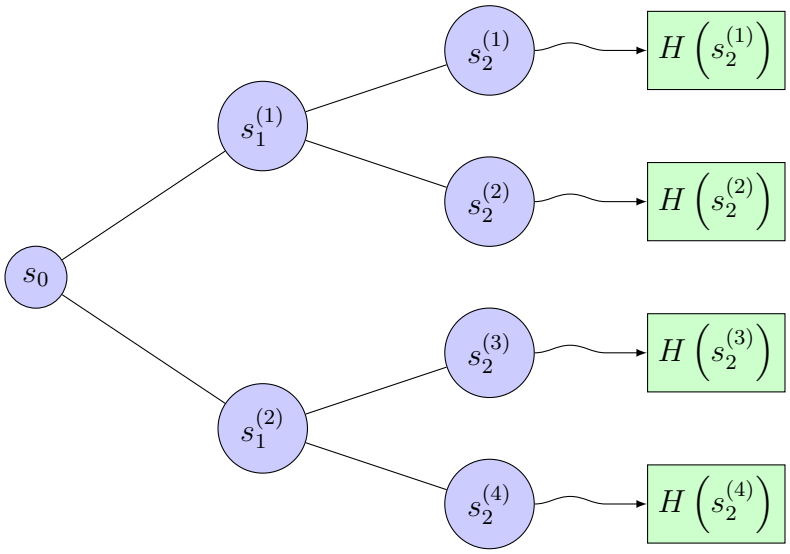


Figure 5.70: Illustration of Model Predictive Control with a Heuristic.

To exemplify how the H-A MPC works, we use the following naïve heuristic on the final observation $\hat{o}_{k_0+N-1}$:

$$H(\hat{o}_{k_0+N-1}) = -\left(\mathrm{SoC}(\hat{o}_{k_0+N-1}) - 0.5\right)^2,$$

where $\mathrm{SoC}(\hat{o}_{k_0+N-1})$ is the state of charge at the final observation $\hat{o}_{k_0+N-1}$.

Intuitively, this heuristic maximizes rewards by aiming for a final SoC close to 50%. This value is set because the agent anticipates (naïvely) that it will need some amount of energy for its actions after the end of the horizon. Figures 5.71 to 5.73 indicatively show sample cases of the strategies planned by the MPC, Stochastic MPC, and Monte Carlo SMPC optimizers using the naïve heuristic. One can observe that optimizer yields a control sequence that results in a final SoC near 0.5, due to the queadratic heuristic. After the lookahead horizon, the model of the system yields a window of trajectories, which can be helpful in calculating a more sophisticated heuristic, described in the following sections.

A more sophisticated H-A MPC algorithm requires a better heuristic. In our approach, explained in Section 4.6, MPC is performed over a finite horizon of N time steps. Then the forecast is extended by generating predictions for an extra interval of L time steps, conditioned on the generated observations of the horizon. These extended forecasts are used to train a LSTM model that estimates the optimal terminal observation, denoted by

$$o^{\mathrm{opt}}_{N+k_0-1} = \mathrm{LSTM}_\theta\Big(\hat{o}^s_{k_0:N+k_0-1},\, \hat{o}^s_{N+k_0:N+k_0+L-1}\Big),$$

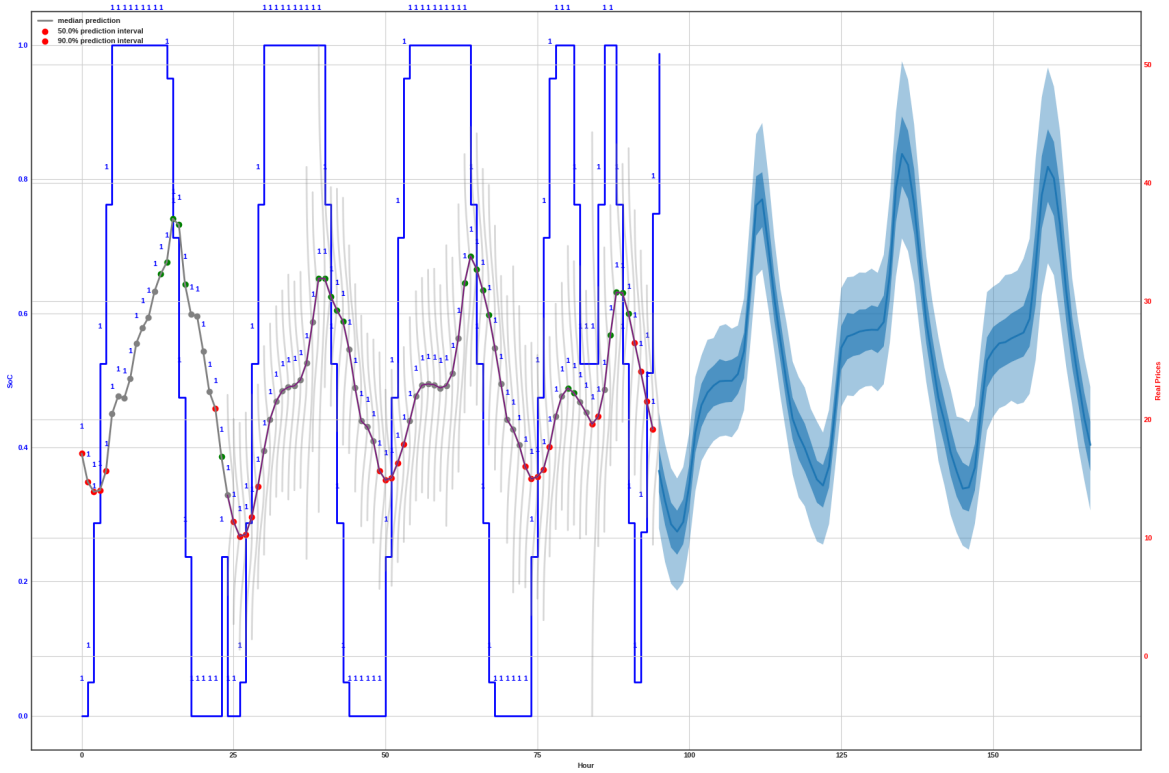


Figure 5.71: Strategy planned by MPC Optimizer, with the predictor extending the optimization horizon.

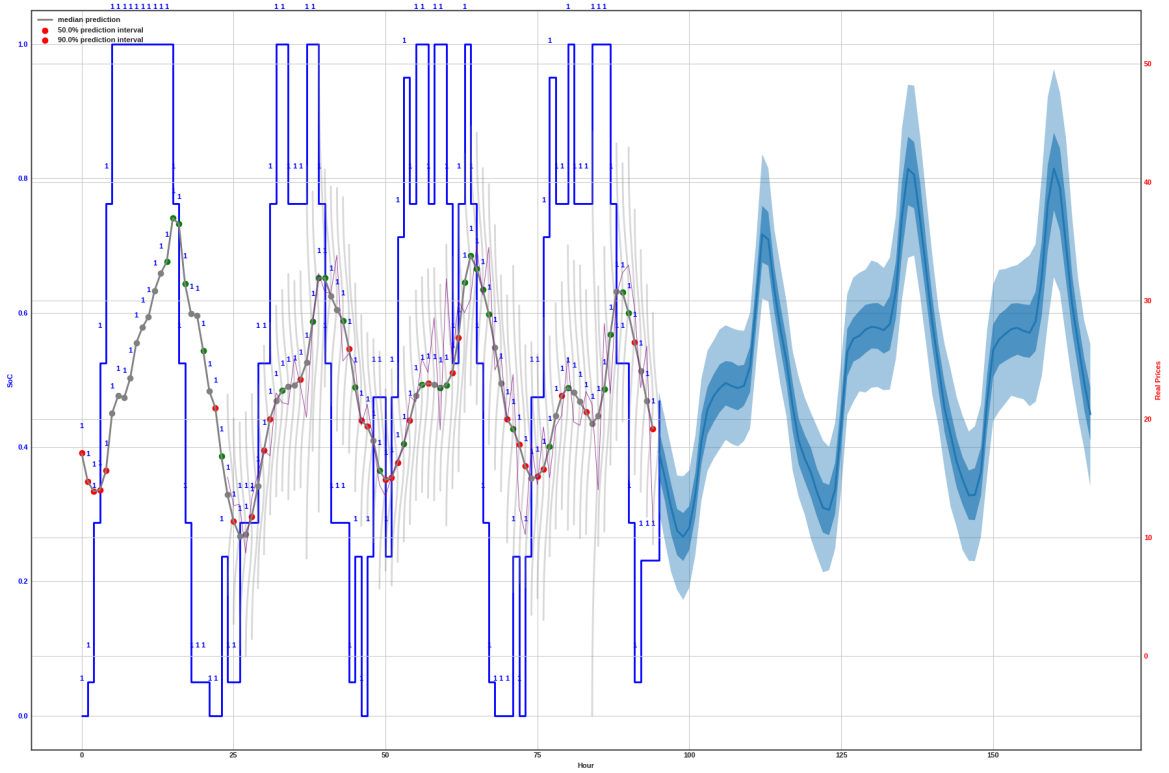


Figure 5.72: Strategy planned by SMPC Optimizer, with the predictor extending the optimization horizon.

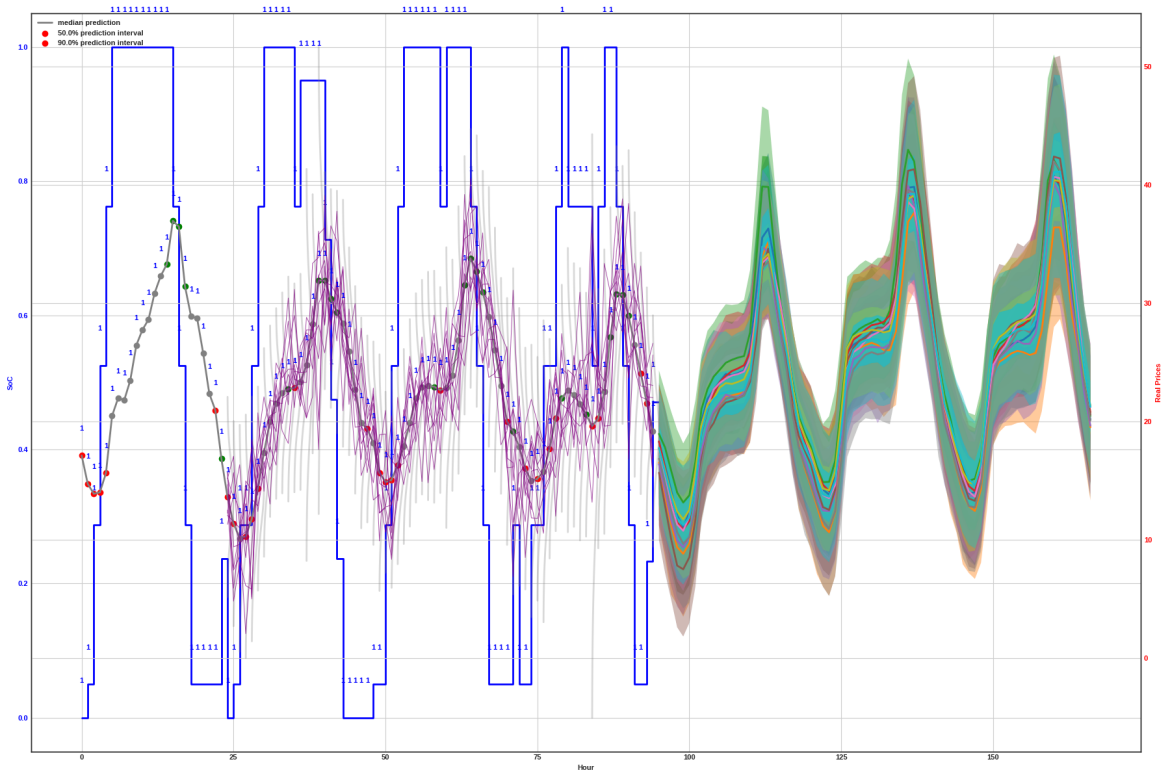


Figure 5.73: Strategy planned by MC-SMPC Optimizer, with the predictor extending the optimization horizon.

where θ are the parameters of the LSTM network. This terminal observation is the observation at step N which the system must approach, in order to yield an optimal decision sequence that accounts for the steps after the end of the horizon.

We implemented the two variants proposed; one with a hard terminal constraint, where the optimization process must end up in the predicted observed state and one with a soft, quadratic constraint, where the MPC algorithm can balance between achieving the terminal state and optimizing the cumulative reward. Figure 5.74 provides an image to compare the predictions made by the LSTM against the actual optimal terminal observed states. One can observe that the LSTM is overall accurate in predicting the optimal SoC for the given sequences, hence provides a good heuristic for the algorithm.

In addition, Figure 5.75 shows the distribution of residuals (differences between actual and predicted values), indicating that the errors are generally small and centered around zero.

Training of the Optimal Terminal State Predictor

The training and testing datasets for the LSTM model contain 16,747 and 4,187 samples, respectively. The LSTM achieved a Mean Absolute Error of 0.0809, Mean Squared Error of 0.0127, and an R^2 score of 0.9149 (meaning that a big part of the variance in the input features is explained by the model). Figure 5.76 show an example of the control strategy that the Heuristic-Augmented version of MPC plans in the market. The agent plans using the forecasts of three days and aims to optimize in a way that the last state of the horizon is the predicted optimal terminal state denoted with a yellow dot (in the case of the hard constraint version) or lies close to it (in the case of the soft constraint version). Once the optimal strategy is calculated, the agent performs

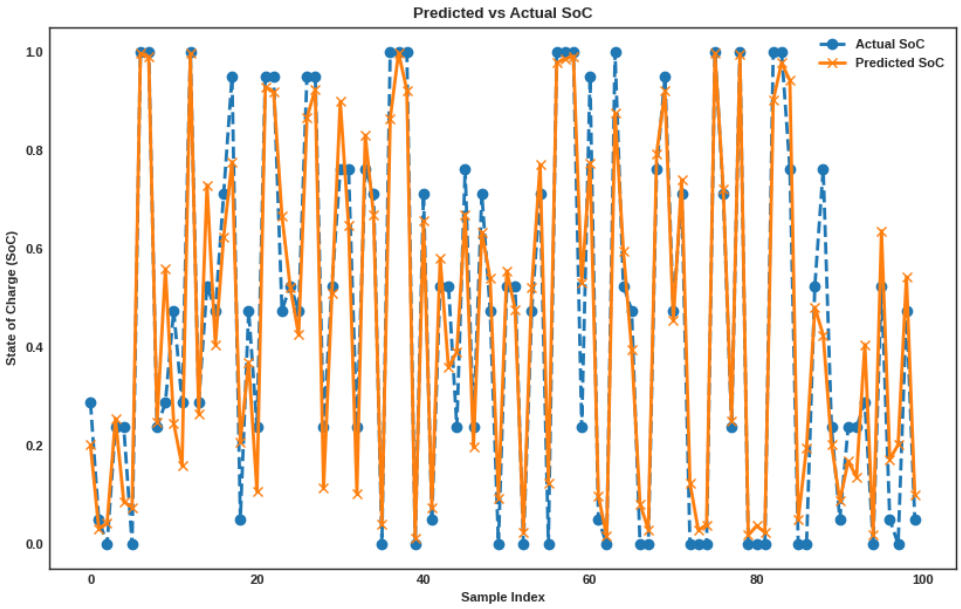


Figure 5.74: Examples of LSTM predictions for optimal SoC to be used as a heuristic by the Heuristic-Augmented MPC algorithm.

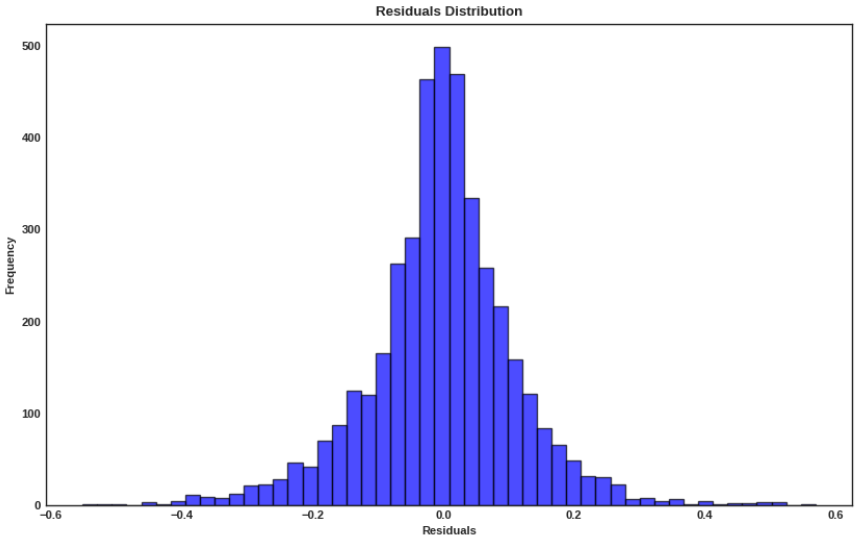


Figure 5.75: Residuals distribution indicating small errors centered around 0.

the best actions for the first day and moves to the next day, where the new prices are published, and more accurate forecasts are generated.

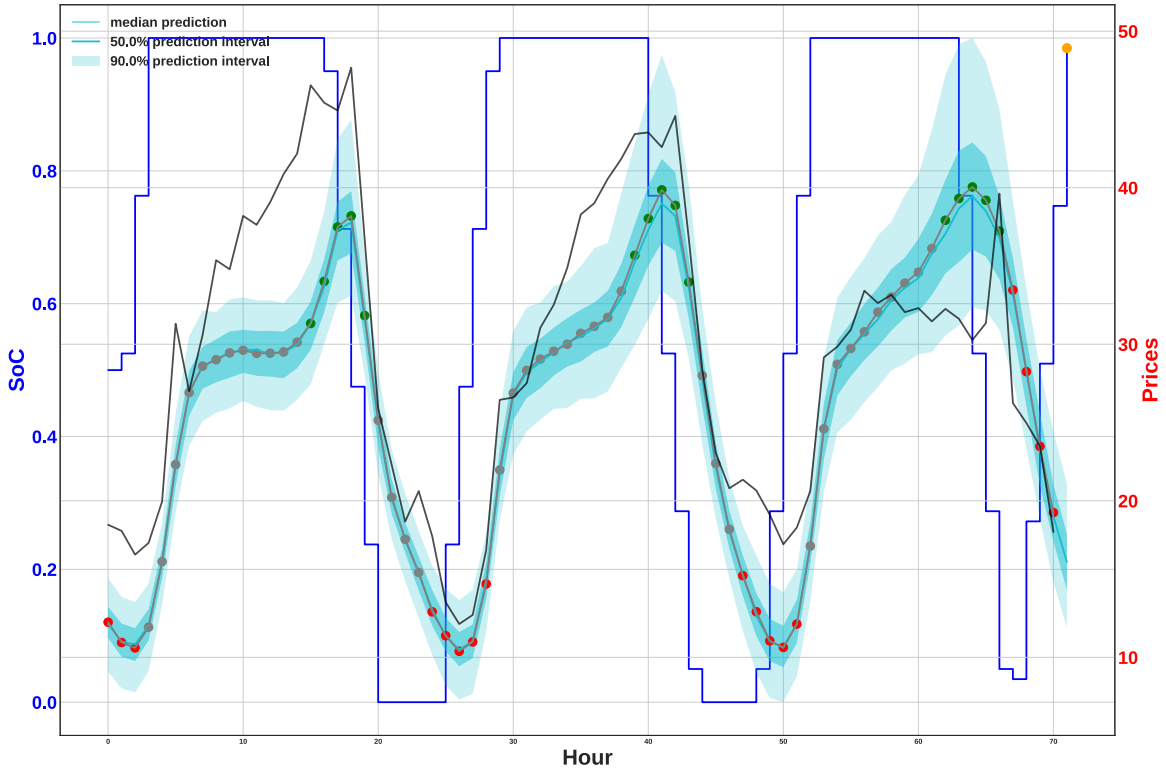


Figure 5.76: Strategy planned by the Heuristic-Augmented MPC Optimizer.

The overall behavior of the Heuristic-Augmented MPC algorithm over a whole month is illustrated in Figures 5.77 and 5.78. As with other implementations of our proposed framework, the agent's behavior proves to be efficient. In fact, the numeric results presented in Section 5.10 show that the MPC augmented with our defined heuristic consistently, albeit slightly, enhances the optimization process of the standard MPC algorithm. This consistent improvement highlights the potential of such an enhancement, suggesting that in other applications, its impact could be even more substantial.

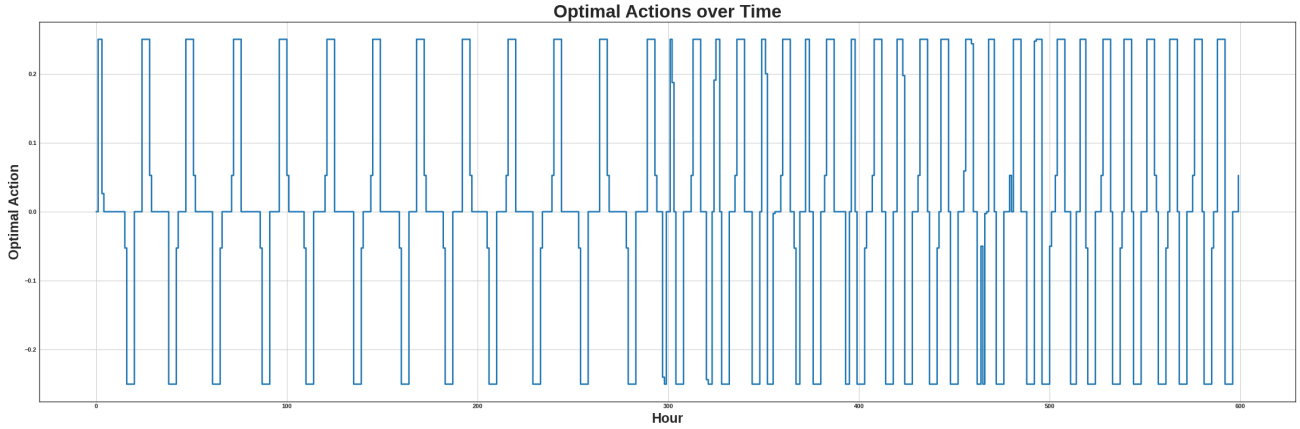


Figure 5.77: Optimal actions returned for a month by the MPC optimizer with heuristics.

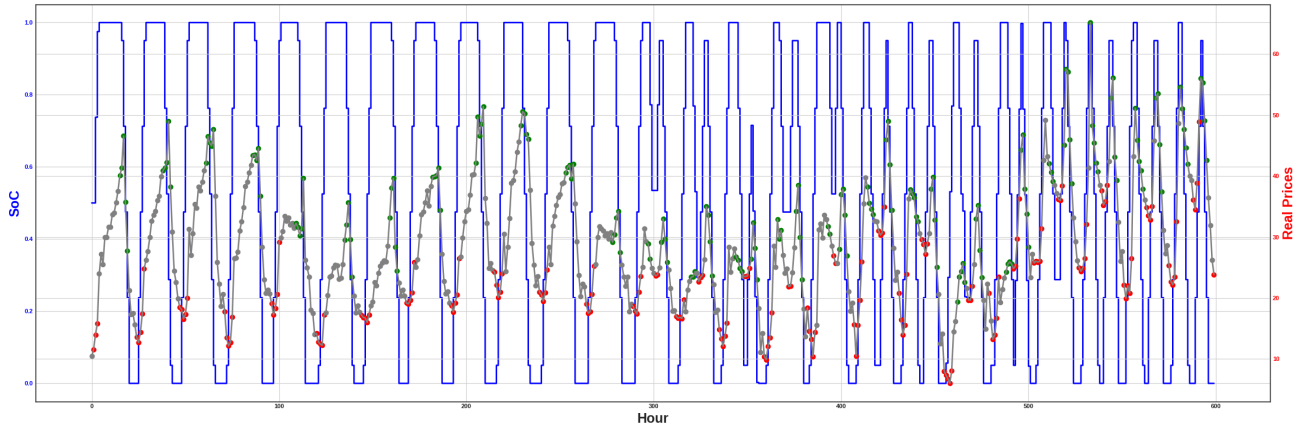


Figure 5.78: SoC levels along with fixed hourly prices for a month after applying the MPC Optimizer with heuristic.

5.10 Diffusion-Informed MPC Variants Comparison

We experimented with all *Diffusion-Informed MPC* variants by applying them in 25 consecutive days, for 10 months and we collected their total rewards in Tables 5.4 to 5.6.

Table 5.4 compares the performance of the MPC variants. In these experiments, we also considered two idealized benchmarks: the Perfect MPC, which has full knowledge of the future price trajectory (infinite horizon), so it yields the absolute best possible control, and the Oracle MPC, which uses the actual prices as its forecast model (a perfect forecaster - or an Oracle that informs about the actual prices in a same horizon as the rest MPC implementations). Our model, *Diffusion-Informed MPC*, gives results that closely match these idealized benchmarks. The average rewards of the Deterministic MPC, Monte Carlo Stochastic MPC and Scenario Tree-Based MPC were found to be 85.78, 86.96, and 85.27 respectively. Comparing with the average rewards of 94.26 for Perfect MPC and 88.93 for Oracle MPC, our method showcases great performance. The gaps of the Deterministic MPC, Monte Carlo Stochastic MPC and Scenario Tree-Based MPC with the Oracle MPC benchmark were merely 3.54%, 2.22%, 3.5%, indicating that a Diffusion-Based forecaster produces sufficiently accurate and detailed forecasts to guide the MPC optimization effectively.

We note that the average reward of the Stochastic MPC version (using one realization) is lower, as expected, however, this reinforces the claim that a diffusion-based forecaster is a powerful model of the system, as even one (noisy) realization of the future trajectory is enough to calculate a good decision sequence. One may compare this method's average reward of 74.99 with the average reward of the best classical MPC method, shown in Table 5.6, which is 61.8, having a gap of 21.34%. Therefore, even with its single sampling, the *D-I MPC* implementation still outperforms other methods, indicating that our approach remains superior for applications demanding rapid optimization.

Furthermore, the incorporation of multiple forecast trajectories in MC SMPC helps to average out the uncertainty over the prediction horizon, yielding more robust decisions. The Scenario Tree-Based MPC variant results in a slightly lower average reward of 85.27, only 1.98% worse, making it still a highly valuable and robust framework. This minor difference in average reward is often a small trade-off for the inherent advantages of the scenario tree approach, particularly its representation of future uncertainties and the non-anticipativity constraints. In situations where the agent has to make cascading decisions and manage the risk, the Scenario Tree-Based MPC version can be superior. Such examples are energy scheduling, smart grid management, financial portfolio optimization, etc.

Additionally, the comparison of MPC with and without heuristic augmentation in Table 5.5 reveals that including a heuristic at the end of the look-ahead horizon can improve performance, although in our current setup the improvement is small. In this application, TimeGrad is already a very powerful forecasting model and the forecasting horizon is big (as the forecasting horizon extends, the terminal state's impact becomes less significant), so the relative improvement from heuristic augmentation is small. However, the consistent gains observed (H-A MPCs yield slightly better results than MPC, in every experiment conducted except one) suggest that in other problem settings, where the forecasting model may not be as precise, or the system dynamics are more chaotic, a well-designed heuristic could have a significant impact.

Month Month	**Perfect MPC**	**Oracle MPC**	**MPC**	**SMPC**	**MC SMPC**	**DST SMPC**	**FC RDST SMPC**	**BH RDST SMPC**
2018-06	99.98	97.13	90.72	73.88	92.11	94.47	**97.57**	96.25
2018-07	147.46	141.56	134.61	119.13	138.75	**140.00**	139.73	139.10
2018-08	123.11	115.31	111.62	95.56	112.43	112.63	**112.99**	112.31
2018-09	108.44	102.32	**100.65**	86.48	100.53	100.20	97.87	98.80
2018-10	104.05	95.36	**103.48**	89.45	102.84	101.17	96.08	96.58
2019-04	60.35	54.62	53.42	47.85	**53.51**	45.00	45.64	46.16
2020-12	68.85	59.31	60.58	49.00	**61.61**	59.92	61.05	58.31
2021-01	64.03	59.00	57.09	53.84	56.60	58.27	**61.05**	56.96
2021-02	109.34	108.33	95.46	87.03	**96.69**	95.71	90.71	88.38
2021-03	56.94	56.35	50.15	47.68	**54.54**	50.72	50.05	49.78
Sum	942.57	889.30	857.78	749.90	**869.62**	858.09	852.74	835.34
Average	94.26	88.93	85.78	74.99	**86.96**	85.81	85.27	83.53

Table 5.4: Comparison of rewards for different MPC methods (higher is better). Columns represent: *Perfect MPC* (idealized benchmark with full future knowledge), *Oracle MPC* (idealized benchmark with perfect forecaster), *Deterministic MPC* (MPC with single point aggregated forecast), *Stochastic MPC* (SMPC with one trajectory realization), *Monte Carlo SMPC* (SMPC with 100 realizations), *Diffusion Scenario Tree SMPC*, *Forward-Clustering Reduced Diffusion Scenario Tree SMPC*, and *Backward-Hierarchical Reduced Diffusion Scenario Tree SMPC*.

Month	**Oracle MPC**	**MPC**	**H-A MPC (Hard Constraint)**	**H-A MPC (Soft Constraint)**
2018-06	97.13	90.72	90.68	**90.97**
2018-07	141.56	**134.61**	134.55	134.33
2018-08	115.31	111.62	**111.62**	111.59
2018-09	102.32	100.65	100.69	**100.83**
2018-10	95.36	103.48	**103.86**	103.75
2019-04	54.62	53.42	53.30	**53.59**
2020-12	59.31	60.58	60.76	**60.88**
2021-01	59.00	57.09	**57.36**	57.21
2021-02	108.33	95.46	95.45	**95.63**
2021-03	56.35	50.15	**50.16**	49.60
Sum	889.30	857.78	**858.40**	858.38
Average	88.93	85.78	**85.84**	85.84

Table 5.5: Comparison of MPC and Heuristic-Augmented MPC methods with hard and soft constraints, relative to Oracle MPC. (higher is better).

5.11 Diffusion vs. Classical Forecasting for MPC

Table 5.6 presents the comparison between the performance of MPC guided by a diffusion-based forecaster (TimeGrad) and MPC guided by several classical forecasting models. We used again the cumulative reward obtained over one-month periods for 10 months. The *Diffusion-Informed MPC* achieves an average reward of 85.78, substantially outperforming the best classical approach (using an autoregressive model) with average reward 61.80. Our method has an advantage of approximately 38.8% over the best classical forecaster method.

Month	**TimeGrad MPC**	**AR MPC**	**ARIMA MPC**	**SARIMA MPC**	**VAR MPC**	**CNN MPC**	**LSTM MPC**
2018-06	**90.72**	87.59	−1.28	80.25	65.84	60.47	74.84
2018-07	**134.61**	109.56	−3.11	106.86	68.81	3.59	38.13
2018-08	**111.62**	93.55	−2.55	93.4	61.02	−15.46	3.59
2018-09	**100.65**	81.44	1.33	83.38	60.77	−1.19	32.56
2018-10	**103.48**	57.71	−1.31	62.07	60.85	38.76	79.75
2019-04	**53.42**	16.45	−1.64	25.34	30.77	18.73	50.13
2020-12	**60.58**	47.05	−0.79	35.27	30.65	12.54	54.41
2021-01	**57.09**	47.51	−0.16	38.39	37.63	−2.23	42.87
2021-02	**95.46**	56.69	−4.39	36.93	56.99	−19.42	−46.61
2021-03	50.15	20.44	−0.92	22.24	28.2	−5.31	**50.88**
Sum	**857.78**	618.00	−14.82	584.14	501.54	90.48	380.54
Average	**85.78**	61.80	−1.48	58.41	50.15	9.05	38.05

Table 5.6: Comparison of rewards for MPC methods using various predictive models (higher is better).

It is important to note that, although the deep learning models we trained achieve better error metrics compared to autoregressive models, their performance in guiding MPC algorithms is not as good. The key reason lies in the specific application on which we test our method, since in energy arbitrage accurately predicting the peaks and valleys of energy prices is far more critical than merely achieving low error rates. The trained CNN and LSTM models tended to estimate the trends of the price time series sometimes with a small shift, resulting in the agent, which follows the policy created by the MPC, being confused on when it is optimal to make certain transactions. This is illustrated in Figure 5.79, where the CNN and LSTM models demonstrate better anticipation of reward compared to classical models, since they provide more accurate forecasts. However, the MPC algorithms based on these deep learning models do not exhibit improved performance, due to the said shifts, and since energy prices are highly time-dependent, autoregressive models inherently provide better temporal correlation. This underscores once again the importance of accurately predicting the details in the state trajectory, hence diffusion-based forecasters are highly suitable to guide the MPC. We observe that all variants of *Diffusion-Informed MPC*, demonstrate much closer alignments between the expected and actual outcomes, reinforcing our method's strength in capturing the intricate dynamics and yielding a robust policy.

Figure 5.80 displays a sample day of the calculated plans for the various classical MPC

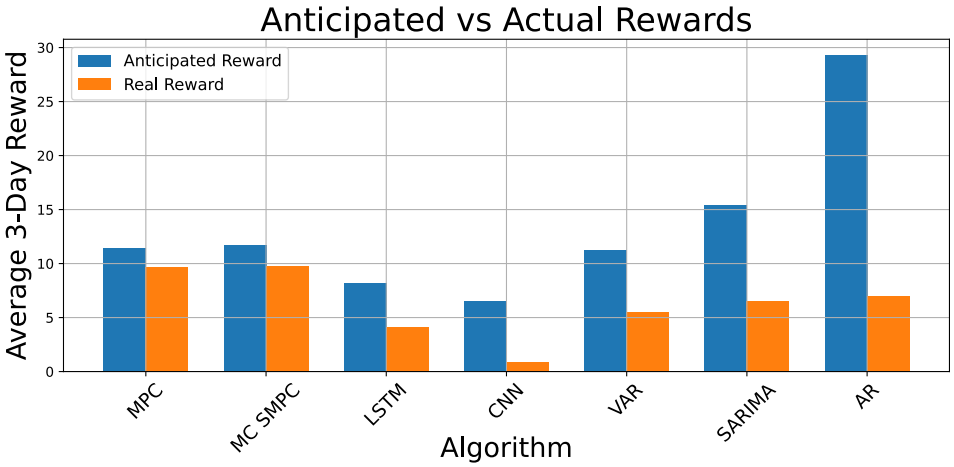


Figure 5.79: Comparison of anticipated and actual rewards (in 3-day windows) for the various implementations, sorted by how closely the anticipated reward matches the actual reward.

methods, compared with *D-I MPC*. These plots highlight how the forecasted price trajectories and the resulting control actions differ among the methods and one may observe several patterns. The AR forecasting model tends to exhibit increasingly large oscillations as predictions progress further from the starting point, meaning that they are not robust enough to plan on. However, the oscillations become apparent after the first 24 hours, which mitigates their influence. To explain further, since MPC is a recending horizon optimizer, it applies the first 24 actions before shifting the horizon window forward, hence the oscillations are too far away to have a significant effect on the policy. The SARIMA model, although also prone to divergence, produces forecasts that follow the overall trend more smoothly; however, its predictions gradually have a growing offset relative to the target series, as error accumulates. The VAR model is more stable than the simple AR model, however it struggles in predicting the minima and maxima which, as explained before, has devastating effects in the calculated policy. Furthermore, the CNN-based forecasting model shows an underestimation of the time series, although it is not a typical behavior; in other windows, its predictions follow the target series well. On the other hand, the LSTM predictions align a lot better with the target. Both the CNN and LSTM models, despite capturing the trend, exhibit small but noticeable offsets in the prediction of the minima and maxima of the series, which significantly affect the decision-making process of the MPC.

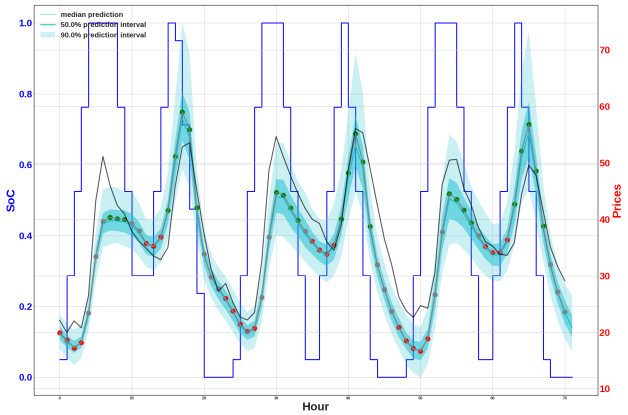
Consequently, diffusion-based forecasters, with their enhanced expressivity and ability to generate detailed trajectories, are highly suitable to guide the MPC, as they produce forecasts that represent the underlying dynamics better and thus lead to superior control performance.

We also compare Scenario Tree-Based MPC plans constructed from classical point-forecast models versus the diffusion forecaster. For each classical model (VAR or LSTM), we fit a Gaussian to its one-step-ahead residuals on historical data and sample scenario trajectories by drawing

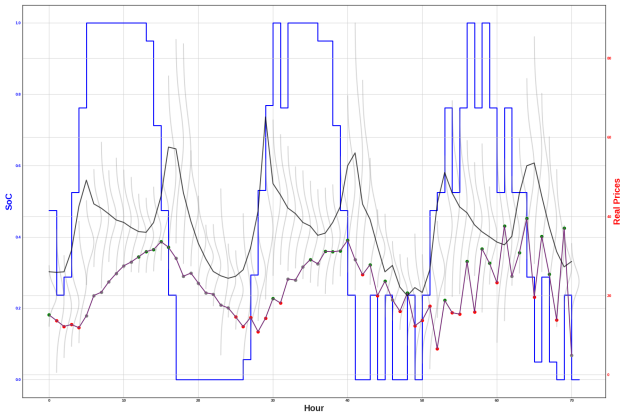
$$\tilde{x}_{k+j}^{(m)} \sim \mathcal{N}\big(\hat{x}_{k+j}, \sigma_{\text{err}}^2\big),$$

where $\hat{x}_{k+j}$ is the point forecast and σ_{err} its residual standard deviation. We then solve a multistage MPC over these sampled trees. For the diffusion forecaster, the tree is sampled directly from its learned conditional diffusion model.

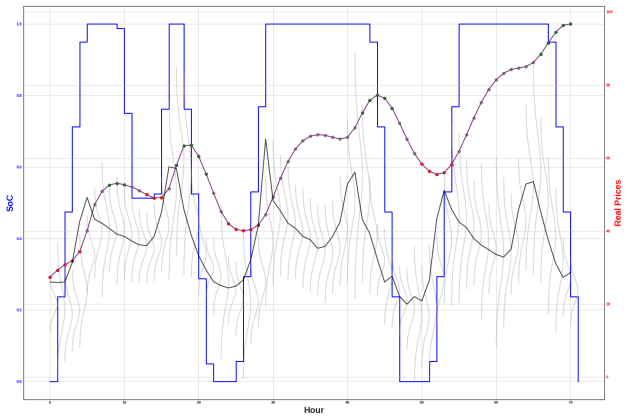
Note that the scenario tree generated by VAR was so noisy and irregular that the MPC solver had difficulty optimizing across all branches. As a result, we pruned the least probable trajectories to ensure tractable and stable control optimization.



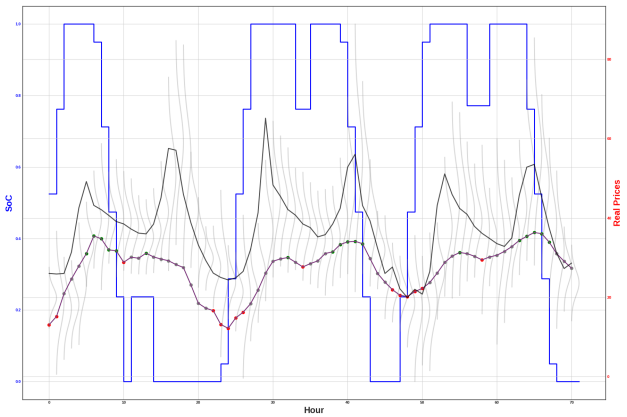
(a) Diffusion-based forecasting model.



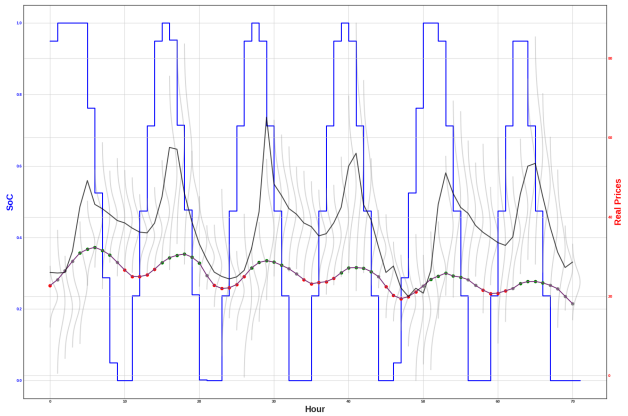
(b) AR forecasting model.



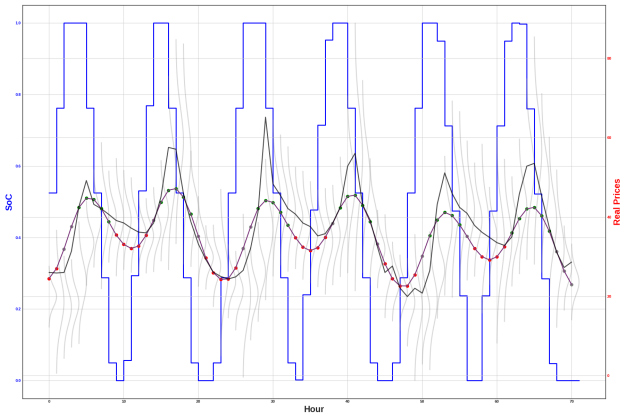
(c) SARIMA forecasting model.



(d) VAR forecasting model.

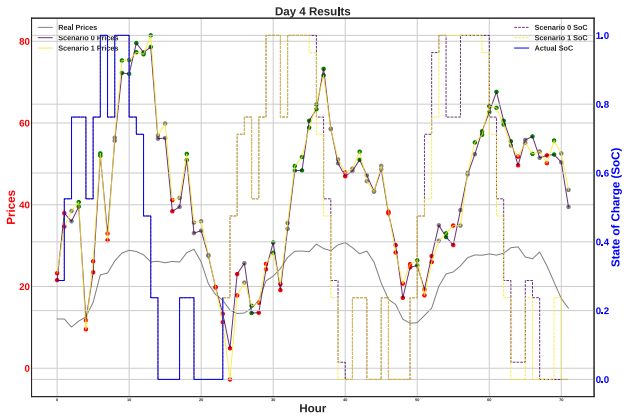


(e) CNN forecasting model.

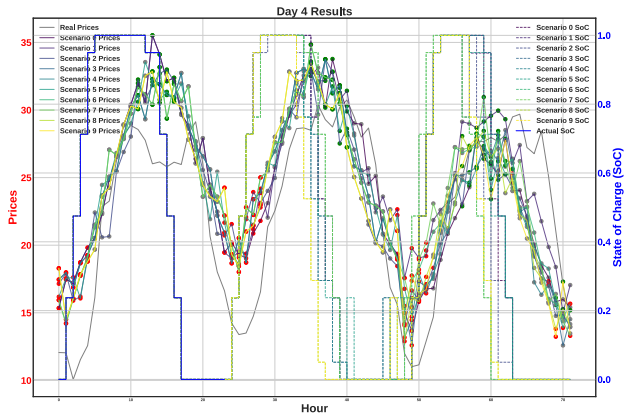


(f) LSTM forecasting model.

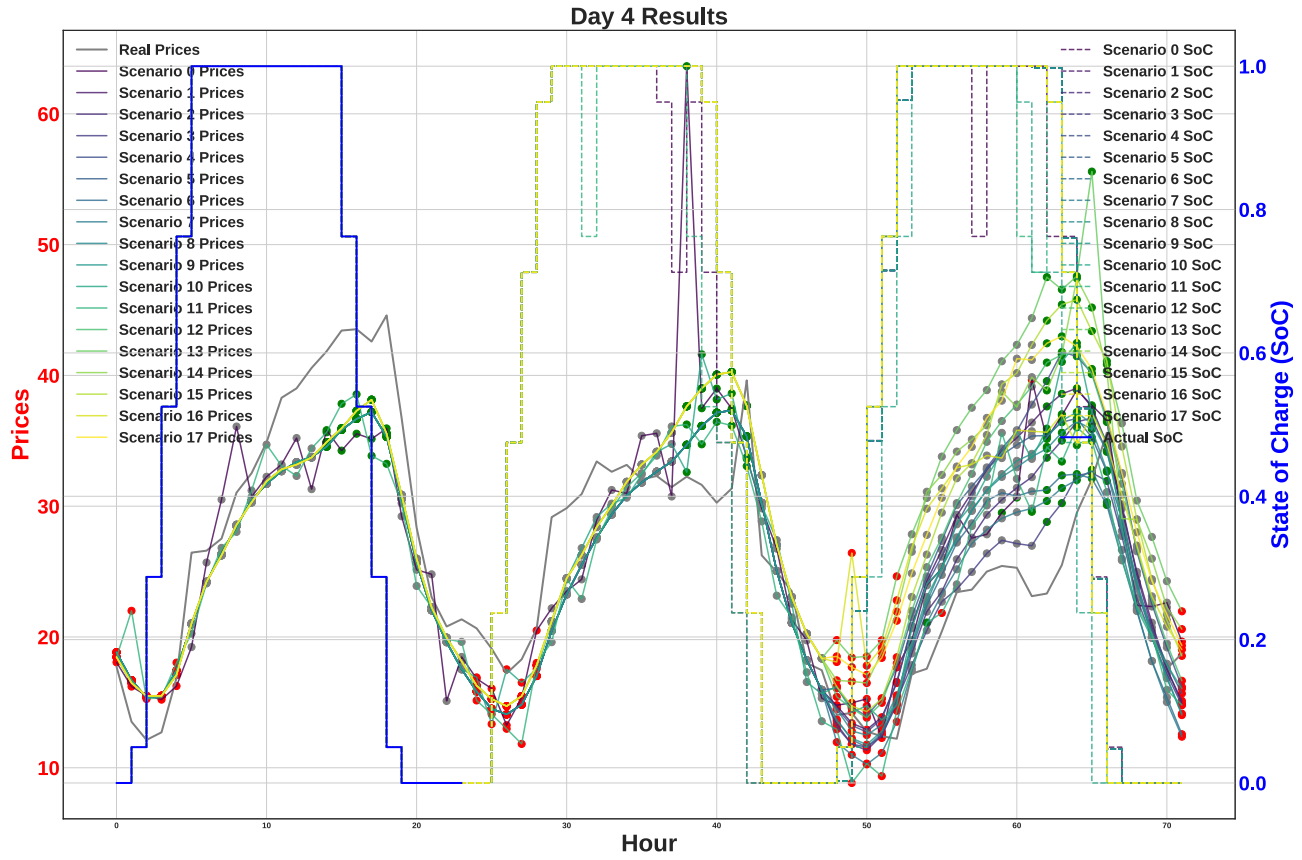
Figure 5.80: Control trajectories for MPC strategies using different forecasting models. Subfigure (a) shows the diffusion-based forecasting model (TimeGrad) guiding the MPC, whereas subfigures (b) through (f) display the corresponding trajectories when using classical forecasting methods: AR, SARIMA, VAR, CNN, and LSTM respectively.



(a) VAR-based tree MPC



(b) LSTM-based tree MPC



(c) Diffusion-based tree MPC

Figure 5.81: Control trajectories under scenario-tree MPC. (a) VAR-based, (b) LSTM-based, and (c) diffusion-based trees. Scenario trees from VAR and LSTM are noisy and produce disturbed control plans, whereas the diffusion-based tree yields coherent scenarios and a robust plan.

5.12 Diffusion-Based MPC vs. Model-Free RL

Table 5.7 compares the performance of our Diffusion-Informed MPC with that of several model-free RL implementations based on various DQN architectures. In the experiments, the model-free methods are provided with perfect knowledge of the next 12 real prices, yet even the best-performing DQN agent yields an average reward of 26.92, which is approximately 69.5% worse than our method. This large difference underscores the advantage of using a model to optimize decisions when an accurate one exists.

Beyond performance, a significant advantage of our method over model-free RL approaches lies in its robustness and the fact that tuning is significantly easier. Once good parameters are found for the diffusion-based forecaster, its predictions remain consistent across all training runs. This consistency in forecasting leads to a robust and stable optimization process within the MPC framework.

In contrast, training model-free RL methods is challenging. The immense size of their parameter space makes it very difficult to search for optimal configurations. Furthermore, given the inherent complexity and stochastic nature of real-world system dynamics, achieving convergence in model-free RL agents often demands a huge number of training iterations. For instance, our best-performing DQN agent required 15,000,000 iterations.

While it is theoretically proven that model-free RL methods, when applied to Markov Decision Processes (and not even POMDPs that usually model real-world systems) and given sufficient training iterations under certain conditions (e.g., proper exploration, suitable function approximation), can eventually approach optimal behavior, the practical implications are often prohibitive. The extensive time and computational complexity required for training make model-free approaches less desirable for applications where a robust and accurate system model, like our diffusion-based forecaster, is available. In such scenarios, the model-based approach offers a far more efficient and reliable path to good agent behavior.

	Model-Free RL (Knowledge: 12 real prices ahead)					**Model-Based RL**
Month	**DQN** $32 \times 64 \times 32$	**DQN** $64 \times 128 \times 64$	**DQN** $128 \times 256 \times 128$	**DQN** $64 \times 128 \times 128 \times 64$	**DQN** $64 \times 128 \times 256 \times 128 \times 64$	**TimeGrad**
2018-06	8.40	30.89	21.34	4.62	61.28	**92.11**
2018-07	14.99	53.44	30.08	8.80	45.05	**138.75**
2018-08	6.14	46.00	10.64	3.56	25.57	**112.43**
2018-09	3.87	28.21	17.99	3.07	20.14	**100.53**
2018-10	8.29	31.75	18.96	8.79	14.46	**102.84**
2019-04	3.43	18.65	10.25	2.41	6.00	**53.51**
2020-12	3.55	15.35	9.32	3.67	7.18	**61.61**
2021-01	2.91	19.37	10.47	7.01	10.62	**56.60**
2021-02	2.58	12.44	14.39	3.36	0.32	**96.69**
2021-03	1.59	13.10	7.46	6.43	5.65	**54.54**
Sum	55.76	269.15	151.80	51.02	196.27	**869.62**
Average	5.58	26.92	15.18	5.10	19.63	**86.96**

Table 5.7: Performance comparison of Model-Free RL with different DQN architectures and Model-Based RL (TimeGrad).

Chapter 6

Conclusion and Discussion

Contents

6.1 Conclusion

This thesis developed a set of novel frameworks for decision-making under uncertainty in stochastic dynamical systems. We integrated state-of-the-art diffusion-based probabilistic forecasting in control algorithms to handle the uncertainty and complexity of real-world multi-dimensional stochastic systems. Our core insight was that diffusion models, which are designed to learn complex data distributions, can be successfully adapted to the time-series domain, to yield rich predictive distributions. On these results, we defined and implemented a collection of Model Predictive Control algorithms to perform deterministic and stochastic, single-stage and multi-stage control, based on the diffusion model's forecasts distribution.

Through extensive experiments in the problem of energy arbitrage, we demonstrate that our method, *Diffusion-Informed MPC* (*D-I MPC*), consistently achieved substantially better metrics when compared to traditional approaches. This outcome is explained by the inherent ability of diffusion models to accurately capture the intricate dynamics of the system. Furthermore, the performance of our *D-I MPC* variants closely matched idealized benchmarks that we implemented, indicating that our method optimized agent actions nearly as effectively as algorithms operating with perfect models of the system.

Our model-based framework demonstrated a distinct advantage over various model-free Reinforcement Learning (RL) implementations too. This substantial difference underscores the benefits of using an accurate model for optimization when it is available. An additional advantage of our approach is the robustness and easier tuning, in contrast with the high complexity of tuning and time-consuming training of model-free RL methods.

Our work has been accepted for publication at EUSIPCO 2025, underscoring the practical value of our method. In sum, these contributions point to a promising direction for applying deep generative models in real-world decision problems.

6.2 Summary of the Contributions of this Thesis

1. **Deterministic MPC with TimeGrad.** We aggregate the diffusion model's forecasts into a single point trajectory and solve a standard receding-horizon optimization, retaining the simplicity of classical MPC while utilizing the expressive forecasts of TimeGrad.

2. **Monte Carlo Stochastic MPC.** We sample multiple full-horizon trajectories from the diffusion-based forecaster and optimize control actions to maximize the *average* reward across these scenarios, approximating the expected reward by embedding uncertainty into the decision process.

3. **Scenario Tree-Based MPC.** By clustering the diffusion model's forecasts at multiple stages, we discretize the probability distribution and organize it in a tree structure allowing planning under non-anticipativity constraints.

4. **Heuristic-Augmented MPC.** To mitigate finite-horizon effects, we train a lightweight optimal terminal state predictor and incorporate its output as terminal constraint in the MPC formulation, implicitly extending the planner's foresight without much additional computational cost.

6.3 Future Work

Building upon the frameworks defined in this thesis, there are several promising areas for future research:

1. **Advanced Scenario Tree Pruning:** While Diffusion Scenario Trees result in a robust optimization process, further research is needed to mitigate the effects of tree trimming. This means to explore dynamic pruning or heuristic-driven concepts that adapt to the generated scenarios and decide which nodes are worth to expand or prune.

2. **Hybrid Reinforcement Learning Architectures:** Another future direction involves creating hybrid RL approaches that combine the strengths of both model-free and model-based techniques. Such an approach would use the explicit knowledge of system dynamics provided by our diffusion-based forecaster, and also incorporate the adaptive capability of model-free RL to refine policies through interaction. Architectures such as Dyna-style algorithms that use our diffusion-based forecaster to generate imagined trajectories for policy evaluation, and Q-learning variants that use the learned model to approximate cost-to-go values beyond the planning horizon could enhance the learning process.

3. **Broader Application Domains:** To provide more solid proof of the efficiency and robustness of our methodology, future work can exploring additional use cases beyond energy arbitrage, such as smart grid control, supply chain management, autonomous driving, or financial portfolio optimization.

4. **Comparison of Various Diffusion Models in MPC:** While TimeGrad provided a powerful forecasting model, it would be interesting to explore and compare the performance of other diffusion models for time series within our MPC framework.

5. **Integration of Anomaly Detection:** Future research could also investigate the incorporation of anomaly detection diffusion models into the framework. This would allow the MPC to plan more accurately and exploit the anomalous fluctuations of the system, such as price spikes or sudden demand shifts.

6. **Incorporation of Event Data**: Another future step should also involve incorporating other sources of data, such as weather information and news feeds. These external factors provide valuable information regarding the unexpected system behaviors, like sudden price spikes, and can be added into the diffusion model as a condition vector on the generation of the future trajectories.

Bibliography

[1] IEEE Spectrum, "AI at Dartmouth: The Historic Workshop That Launched an Industry." https://spectrum.ieee.org/dartmouth-ai-workshop. Accessed: 2025-05-27.

[2] S. E. Community, "Architecture neural network with weights." Tex StackExchange: Neural Network Architecture, 2019. Accessed: 2025-01-12.

[3] T. Tantau, "Neural Networks with TikZ." TikZ.net Neural Networks Tutorial, 2023. Accessed: 2024-07-05.

[4] H. Iqbal, "PlotNeuralNet." https://github.com/HarisIqbal88/PlotNeuralNet. Accessed on: May 25, 2025.

[5] T. S. User, "How do I draw an LSTM cell in TikZ?." Tex StackExchange: LSTM Cell, 2018. Accessed: 2025-01-12.

[6] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-Based Generative Modeling through Stochastic Differential Equations," in *Proceedings of the International Conference on Learning Representations (ICLR 2021)*, 2021. Oral presentation; arXiv preprint arXiv:2011.13456.

[7] K. Rasul, C. Seward, I. Schuster, and R. Vollgraf, "Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting," in *International conference on machine learning*, pp. 8857–8868, PMLR, 2021.

[8] T. Yan, H. Zhang, T. Zhou, Y. Zhan, and Y. Xia, "ScoreGrad: Multivariate Probabilistic Time Series Forecasting with Continuous Energy-based Generative Models," 2021.

[9] S. Zarifis, I. Kordonis, and P. Maragos, "Diffusion-Based Forecasting for Uncertainty-Aware Model Predictive Control," 2025.

[10] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," 1986.

[13] M. I. Jordan, "Chapter 25 - Serial Order: A Parallel Distributed Processing Approach," in *Neural-Network Models of Cognition* (J. W. Donahoe and V. Packard Dorsel, eds.), vol. 121 of *Advances in Psychology*, pp. 471–495, North-Holland, 1997.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pp. 353–374, 2023.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," 2023.

[17] K. P. Murphy, *Probabilistic Machine Learning: Advanced Topics*. MIT Press, 2023.

[18] D. Reynolds, *Gaussian Mixture Models*, pp. 659–663. Boston, MA: Springer US, 2009.

[19] C. Sammut and G. I. Webb, eds., *Expectation-Maximization Algorithm*, pp. 387–387. Boston, MA: Springer US, 2010.

[20] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[21] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations (ICLR 2014), Conference Track Proceedings*, 2014. arXiv preprint arXiv:1312.6114.

[22] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine Learning*, ICML 2008, (New York, NY, USA), p. 1096–1103, Association for Computing Machinery, 2008.

[23] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[24] D. Rezende and S. Mohamed, "Variational Inference with Normalizing Flows," in *International conference on machine learning*, pp. 1530–1538, PMLR, 2015.

[25] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series Generative Adversarial Networks," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[26] K. E. Smith and A. O. Smith, "Conditional GAN for timeseries generation," 2020.

[27] O. Fabius and J. R. Van Amersfoort, "Variational Recurrent Auto-encoders," *arXiv preprint arXiv:1412.6581*, 2014.

[28] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep Unsupervised Learning using Nonequilibrium Thermodynamics," *arXiv preprint arXiv:1503.03585*, 2015.

[29] J. Ho, A. Jain, and P. Abbeel, "Denoising Diffusion Probabilistic Models," *arXiv preprint arXiv:2006.11239*, 2020.

[30] L. Lin, Z. Li, R. Li, X. Li, and J. Gao, "Diffusion Models for Time Series Applications: A Survey," *Frontiers of Information Technology & Electronic Engineering*, vol. 25, no. 1, pp. 19–41, 2024.

[31] Y. Li, X. Lu, Y. Wang, and D. Dou, "Generative Time Series Forecasting with Diffusion, Denoise, and Disentanglement," *Advances in Neural Information Processing Systems*, vol. 35, pp. 23009–23022, 2022.

[32] V. De Bortoli, J. Thornton, J. Heng, and A. Doucet, "Diffusion Schrödinger Bridge with applications to score-based generative modeling," *Advances in Neural Information Processing Systems*, vol. 34, pp. 17695–17709, 2021.

[33] H. Wen, Y. Lin, Y. Xia, H. Wan, Q. Wen, R. Zimmermann, and Y. Liang, "DiffSTG: Probabilistic Spatio-Temporal Graph Forecasting with Denoising Diffusion Models," 2024.

[34] R. Li, X. Li, S. Gao, S. T. B. Choy, and J. Gao, "Graph Convolution Recurrent Denoising Diffusion Model for Multivariate Probabilistic Temporal Forecasting," in *Advanced Data Mining and Applications* (X. Yang, H. Suhartanto, G. Wang, B. Wang, J. Jiang, B. Li, H. Zhu, and N. Cui, eds.), (Cham), pp. 661–676, Springer Nature Switzerland, 2023.

[35] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[36] R. S. Sutton, "Learning to predict by the method of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[37] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[38] R. E. Bellman, *Dynamic programming*. Princeton University Press, 1957.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019.

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," 2017.

[41] D. Kumar, "Reinforcement Learning for Energy Storage Arbitrage in the Day-Ahead and Real-Time Markets with Accurate Li-Ion Battery Dynamics Model," master's thesis, Massachusetts Institute of Technology, 2021.

[42] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, 1959.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[44] C. Chen, X. Lin, and G. Terejanu, "An Approximate Bayesian Long Short-Term Memory Algorithm for Outlier Detection," 2019.

[45] J. Berrisch and F. Ziel, "CRPS learning," *Journal of Econometrics*, vol. 237, p. 105221, Dec. 2023.

[46] L. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[47] Leonard E. Baum and Ted Petrie and George Soules and Norman Weiss, "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164 – 171, 1970.

[48] F. Farnia and A. Ozdaglar, "Do GANs always have Nash equilibria?," in *Proceedings of the 37th International Conference on Machine Learning* (H. D. III and A. Singh, eds.), vol. 119 of *Proceedings of Machine Learning Research*, pp. 3029–3039, PMLR, 13–18 Jul 2020.

[49] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "WaveNet: A Generative Model for Raw Audio," 2016.

[50] Z. Kong, W. Ping, J. Huang, K. Zhao, and B. Catanzaro, "DiffWave: A Versatile Diffusion Model for Audio Synthesis," in *Proceedings of the International Conference on Learning Representations (ICLR 2021)*, 2021. Oral presentation.

[51] E. L. Thorndike, *Animal intelligence: Experimental studies*. Macmillan, 1911.

[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Senior, K. Kavukcuoglu, D. Hassabis, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[53] J. N. Tsitsiklis and B. V. Roy, "Analysis of Temporal-Difference Learning with Function Approximation," *Machine Learning*, vol. 29, no. 1, pp. 101–130, 1997.

[54] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3-4, pp. 285–294, 1933.

[55] New York Independent System Operator, "NYISO Website." https://www.nyiso.com/.

[56] K. Divya and J. Østergaard, "Battery energy storage technology for power systems—An overview," *Electric Power Systems Research*, vol. 79, no. 4, pp. 511–520, 2009.

[57] N. Y. I. S. Operator, "EXPLAINER: Impact of National & Global Conditions on Electricity Prices in New York," 2022. Accessed: 15 June 2025.