Universitat de Lleida

TREBALL DE FI DE GRAU

ESCOLA
POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA
INSPIRING THE FUTURE

Estudiant: Ramon Segarra Riu

Titulació: Grau en Enginyeria Informàtica

Títol de Treball de Fi de Grau: Desenvolupament d'un sistema de control basat en Machine Learning per a un sistema de generació de fred basat en el RCE.

Director/a: Josep Lluis Lerida Monso / Vitor Luiz da Silva

Presentació

Mes: Juliol

Any: 2025

Index of Contents

1 Introducció **8**

1.1 Introducció . . . 8
1.2 Context . . . 8
1.3 Objectius . . . 9
1.4 Estat de l'art . . . 10
1.5 Enfocament i metodologia . . . 11
1.6 Estructura del treball . . . 13

2 Marc teòric **15**

2.1 Funcionament RCE . . . 15
2.2 Xarxes neuronals i Machine Learning . . . 15
2.3 Llenguatge de Programació . . . 17
2.4 Comunicació basada en sol·licituds . . . 17
2.5 Aplicació API . . . 18
2.6 Sistema de control . . . 19

3 Disseny del sistema de control **20**

3.1 Requeriments del sistema . . . 20
3.1.1 Requeriments funcionals . . . 20
3.1.2 Requeriments no funcionals . . . 20
3.1.3 Requeriments de software . . . 20
3.2 Disseny de la IA . . . 21
3.2.1 Elecció del model . . . 21
3.2.2 Preprocessament de les dades . . . 21
3.3 Integració amb el sistema de control . . . 22
3.3.1 Diagrama de seqüència . . . 22
3.3.2 Tipus de API . . . 23
3.3.3 Recepció de les dades . . . 24
3.3.4 Predicció pluja . . . 25
3.3.5 Predicció temperatura . . . 26
3.3.6 Predicció demanda . . . 26
3.3.7 Predicció producció . . . 27
3.3.8 Ajuntar les dades . . . 27
3.4 Disseny del sistema de control . . . 28
3.4.1 Diagrama de l'arquitectura del SC . . . 28
3.4.2 Lògica Sistema de Control . . . 28
3.4.3 Diagrama de seqüència . . . 29

3.4.4 Obrir/Tancar màquina . . . 30
3.4.5 Acció en un time_step . . . 30

4 Implementació del sistema de control **31**
4.1 Desenvolupament del model d'IA . . . 31
4.1.1 Lectura de les dades d'entrenament . . . 31
4.1.2 Preprocessament de les dades . . . 31
4.1.3 Construcció del model . . . 33
4.1.4 Entrenament del model . . . 35
4.1.5 Validació model . . . 36
4.2 Integració de la IA amb el sistema de control . . . 37
4.2.1 Desenvolupament de la API . . . 37
4.2.2 Predicció pluja . . . 37
4.2.3 Predicció temperatura . . . 39
4.2.4 Predicció demanda . . . 42
4.2.5 Predicció producció . . . 43
4.2.6 Retorn de la API . . . 45
4.3 Desenvolupament del sistema de control . . . 46
4.3.1 Comentaris previs . . . 46
4.3.2 Lectura dades . . . 46
4.3.3 Elecció Finestra Temperatura Calenta . . . 47
4.3.4 Elecció Finestra Temperatura Freda . . . 48
4.3.5 Decisions . . . 48
4.3.6 Ajustar prediccions . . . 49

5 Experimentació i resultats **51**
5.1 Metodologia . . . 51
5.1.1 Model IA . . . 51
5.1.2 API . . . 51
5.1.3 Sistema de Control . . . 51
5.2 Resultats . . . 52
5.2.1 Model IA . . . 52
5.2.2 API . . . 57
5.2.3 Sistema de Control . . . 61
5.3 Discussió resultats . . . 62

6 Conclusions **63**
6.1 Conclusions finals . . . 63
6.2 Limitacions . . . 63

6.3 Línies de recerca futures 64

7 Bibliografia **65**

A Codi font Sistema RCE - Training **67**

A.1 Fitxers reutilitzats 67

List of Figures

1 Diagrama de gantt 12
2 Diagrama de seqüència UML de la API 22
3 Diagrama de components 28
4 Exemple selecció finestra 28
5 Diagrama seqüència Sistema de Control 29
6 Desplaçament de la finestra temporal amb predicció autoregressiva 42
7 Comportament temperatura en 2 hores 52
8 Comportament temperatura en 6 hores 53
9 Comportament temperatura en 24 hores 53
10 Comportament temperatura en 48 hores 54
11 Predicció Estiu 55
12 Predicció Tardor 55
13 Predicció Hivern 56
14 Predicció Primavera 56
15 Request URL 57
16 Request body 57

Resum

En aquest treball, es descriurà el disseny i implementació d'un sistema de control intel·ligent per a una màquina basada en la recuperació de calor o Radiative Cooling and Solar Heating (RCE). Per a fer això, el controlador estarà dividit en tres parts.

El Sistema de Control serà l'encarregat de prendre les decisions sobre com haurà d'actuar la màquina. Per a poder decidir què fer, es faran prediccions sobre les condicions de la màquina mitjançant una API. Aquesta permetrà aconseguir 4 tipus de prediccions diferents. Per cadascuna s'usarà una IA pròpia, una API externa, un històric de dades o una fórmula matemàtica.

El sistema de control sempre estarà en funcionament per a poder prendre decisions. Cada 5 minuts, aquest comprovara si s'ha d'obrir o no la màquina, i en quin mode s'haurà d'obrir.

Llavors, el sistema en general farà els seguents passos: Recollirà les dade per a poder fer prediccions a futur, les procsarà i les enviarà a la APi per a que faci les predicicons. Quan reb una resposta de la API prendra la corresponent decisió i s'esperarà els 5 minuts per tornar a fer el mateix cicle.

Resumen

En este trabajo, se describirá el diseño e implementación de un sistema de control inteligente para una máquina basada en la recuperación de calor o Radiative Cooling and Solar Heating (RCE).Para ello, el controlador estará dividido en tres partes.

El Sistema de Control será el encargado de tomar las decisiones sobre cómo deberá actuar la máquina. Para poder decidir qué hacer, se realizarán predicciones sobre las condiciones de la máquina mediante una API. Esta permitirá obtener 4 tipos de predicciones diferentes. Para cada una se usará una IA propia, una API externa, un histórico de datos o una fórmula matemática.

El sistema de control estará siempre en funcionamiento para poder tomar decisiones. Cada 5 minutos, este comprobará si se debe abrir o no la máquina, y en qué modo deberá abrirse.

Entonces, el sistema en general realizará los siguientes pasos: recogerá los datos para poder hacer predicciones a futuro, los procesará y los enviará a la API para que haga las predicciones. Cuando reciba una respuesta de la API, tomará la correspondiente decisión y esperará los 5 minutos para volver a realizar el mismo ciclo.

Summary

This work describes the design and implementation of an intelligent control system for a machine based on heat recovery or Radiative Cooling and Solar Heating (RCE). To achieve this, the controller will be divided into three parts.

The Control System will be responsible for making decisions on how the machine should operate. In order to decide what to do, predictions about the machine's conditions will be made through an API. This API will provide four different types of predictions. For each one, a specific AI, an external API, a historical dataset, or a mathematical formula will be used.

The control system will always be running in order to make decisions. Every 5 minutes, it will check whether the machine should open or not, and in which mode it should operate.

Then, the overall system will follow these steps: it will collect data to make future predictions, process them, and send them to the API so it can generate the predictions. Once it receives a response from the API, it will make the corresponding decision and wait 5 minutes before repeating the same cycle.

1 Introducció

1.1 Introducció

Cada cop més, la societat exigeix sistemes energètics més eficients, ja que hi ha un greu problema en la sostenibilitat energètica i les energies renovables. A més a més, la demanda energètica es cada cop més gran. Avui dia, els edificis arriben a consumir un 40% de l'energia total que es gasta. A Espanya, un 13,9% de l'energia generada és de les renovables, és a dir, un 86,1% de l'energia actual es generada per recursos fòssils [2]. Una proposta de sistema que usa energies renovables és un sistema RCE, que aprofita l'energia solar que és totalment renovable. Aquest aprofitarà la diferència de temperatura entre el dia i la nit per emmagatzemar o bé temperatures calentes o bé fredes.

Aquest tipus d'energia té molts avantatges, però té un greu inconvenient. Com que farà servir l'energia solar, significa que els sistemes RCE tindran una gran dependència de les condicions climàtiques. Si aquestes canvien, el sistema tardarà en reaccionar. O bé generarem energia quan no cal, o bé no generarem en un moment en el que si s'hauria de fer i ens adonem tard.

Per tal d'optimitzar el funcionament, es pot desenvolupar un sistema de control intel·ligent, és a dir, amb una intel·ligència artificial integrada. Amb aquesta és podrà predir el futur per a prendre decisions.

Llavors, amb un sistema basat en RCE i un sistema de control intel·ligent, es pot arribar a tenir un nou tipus de font d'energia renovable.

Aquest treball vol minimitzar aquest inconvenient amb el desenvolupament d'un sistema de control intel·ligent per una màquina basada en el RCE. Mitjançant tècniques de programació d'intel·ligència artificial com les xarxes neuronals, es voldrà predir la situació climàtica i de la màquina futura per a anticipar-se i prendre les decisions òptimes.

Mitjançant les prediccions, la màquina es podrà adaptar abans que passi qualsevol problema i maximitzar la potència generada envers l'energia que es consumeix.

1.2 Context

Per arribar a desenvolupar correctament el sistema, serà necessari diferents moduls.

Per a la lectura de dades, s'ha de tenir en compte d'on es llegeixen. Actualment, la lectura es fa mitjançant arxius amb un històric de dades. Això, però no és el resultat final, ja que en el prototip haurem de connectar el Sistema de Control a una font de dades, ja sigui als sensors directament del PLC o als arxius generats que enregistren els valors dels sensors.

Llavors, un cop llegides les dades, s'hauran de fer les diferents prediccions. N'hi haurà de 4 tipus:

- **Condicions meteorològiques**: En aquesta aconseguirem una predicció del dia de l'índex de pluja. Seleccionarem el temps en els moments més crítics del dia, com la sortida i posta de sol, migdia i mitjanit. Per a fer-ho, cridarem a una API externa.
- **Temperatura dels tancs**: Farem una predicció en 2 hores d'antelació dels tancs fred i calent. Per a fer-ho, mitjançant les dades dels sensors i un model d'IA propi ja entrenat, aconseguirem la temperatura dels tancs en un futur.
- **Demanda**: Aquesta predicció ens permetrà conèixer quina serà la demanda en el futur. Per aconseguir la demanda, actualment s'està llegint directament d'un històric de dades, per tant, aquesta serà teòrica. Aquesta, però, no serà realista, ja que és un històric de dades que es va calcular en un any, cosa que comporta que si les condicions canvien, no s'adaptarà al valor real.
- **Producció**: Finalment, predient la producció aconseguirem saber quan podrà generar la màquina per conèixer si es podrà satisfer la demanda. Aquesta predicció serà analítica, és a dir, es farà servir una fórmula matemàtica que aproximarà la potència. Però per això necessitem saber diferents valors, com la densitat de l'aigua (RHO) o la capacitat calòrica específica (CP). Finalment, necessitarem saber la temperatura del futur, que és la que hem aconseguit a la predicció de la temperatura dels tancs.

Aquestes dades les ha de fer servir el Sistema de Control. Caldrà dissenyar una API per realitzar la connexió i integrar aquesta al Sistema de Control. Per transmetre tota la informació entre els diferents components es farà servir un fitxer JSON. Aquest tipus de document serveix simplement per guardar dades, però de tal forma que la seva manipulació i lectura és fàcil i ràpida.

1.3 Objectius

Aquest treball té com a finalitat desenvolupar un sistema de control intel·ligent per a una màquina basada en el RCE. Per a poder assolir aquest objectiu general, s'han marcat diversos objectius específics per concretar com es farà aquest desenvolupament:

- Dissenyar l'arquitectura necessària per implementar un Sistema de Control intel·ligent alimentat de diferents fonts de dades. Aquest sistema ha de ser modular, eficient i reutilitzable.
- Desenvolupar diferents model de IA per poder fer prediccions futures sobre la temperatura del tancs del sistema RCE, condicions meteorològiques, etc...
- Disseny d'una API per tal de poder consular les prediccions als models intel·ligents
- Integració en el sistema de control de de les diferents fonts d'informació a través de les diferents APIs
- Afegir la lògica de control i validar l'eficàcia del SC amb condicions teòriques conegudes.

1.4 Estat de l'art

En els darrers anys, la intel·ligència artificial i l'aprenentatge automàtic s'han convertit en una eina indispensable per a molts sistemes informàtics. Un d'aquests són els sistemes de control. Gràcies a les IA es pot arribar a analitzar grans volums de dades, cosa que permet optimitzar el funcionament dels dispositius.

Les IA són molt més potents del que ens podem arribar a imaginar, ja que es poden aplicar no només en l'àmbit de dispositius, sinó també en sistemes complets, com xarxes de calefacció en una urbanització sencera [17]. En els casos més complexos, es poden fer diferents prediccions com la de demanda, que permet controlar el sistema d'una forma més eficient.

Les tècniques de *Deep Learning* són especialment útils per a fer prediccions amb dades temporals, mostrant millors resultats on hi ha més variabilitat de dades [18]. A més a més, existeixen mètodes com el Deep Reinforcement Learning (DRL) que permet entrenar perquè el rendiment sigui el més bo possible [19].

Abans de començar a exposar l'estat de l'art, s'ha de nomenar que, per la realització del sistema de control, s'ha utilitzat un model de predicció d'un treball previ. Aquest treball s'anomenava "*Proyecto RCE - Training*", realitzat per *Christian López Garcia* [21]. A més a més, aquest treball s'ha basat en el projecte "INTERFÍCIE DE COMUNICACIÓ ENTRE UN MODEL DE XARXA NEURONAL LÒGIC PREDICTIU I UN PLC EN UN PROTOTIPUS REAL" realitzat per Albert Joan Santacana Gabella [22], amb el que pot ajudar-me a entendre com fer la comunicació entre el model i la PLC. El codi utilitzat està referenciat a l'Annex A.

Actualment, el grup de recerca del CREA està duent a terme proves en un Sistema de Control sense un model d'IA. Els resultats no són dolents, però quan hi ha canvis de clima, el prototip no rendeix tan bé com tocaria. Han buscat noves formes per millorar el rendiment i van fer estudis sobre l'eficàcia de l'ús de models d'IA per predir el futur. Llavors, es vol provar en un entorn real un Sistema de Control intel·ligent. Mantenint el circuit en el qual hi ha el sistema actual, se'n vol afegir un de secundari que es pugui adaptar mitjançant prediccions.

En aquest projecte es va realitzar la fase d'entrenament del model i una versió del model predictiu. Per fer l'entrenament, es va fer un estudi previ del sistema i de les variables se'n va extreure les més importants per a fer una millor predicció.

Per una banda, tenim el component principal del programari, que serà el sistema de control. En el cas d'aquest projecte es fa servir codi font per a declarar el comportament d'aquest. El sistema de control serà l'encarregat de definir com s'ha de comportar la màquina en tot moment, amb relació a uns valors que s'hagin declarat prèviament.

Però aquest sistema de control serà intel·ligent, és a dir, hi haurà integrat un model d'intel·ligència artificial. La intel·ligència artificial és una màquina que és capaç de fer tasques que poden fer

els humans. Malgrat això, serà indispensable tenir una base d'entrenament, perquè aquest pugui arribar a aprendre els patrons en les dades per a poder predir.

Encara que tinguem un model d'IA i el sistema de control, falta tenir una forma per connectar-los. Per a tenir un codi distribuït, es farà servir una aplicació API. Aquesta permet la connexió d'una aplicació amb un codi extern sense tenir accés directe dins del codi. Es podria comprovar una API amb el funcionament d'una caixa negra, on inserint unes dades, et retorna unes altres sense saber el que passa per dins.

A l'hora de desenvolupar el sistema de control, l'API i el model d'IA, no és necessari fer un desenvolupament des de zero. Existeixen biblioteques ja creades que faciliten, en gran manera, desenvolupar nous tipus d'aplicacions.

- Per la part dels sistemes de control, existeixen tecnologies com els PLC, que automatitzen processos industrials de forma automàtica o els sistemes SCADA, que permeten controlar de forma remota sistemes industrials.
- Per la part de les aplicacions API, hi ha per exemple les llibreries Flask o FastAPI que permeten un desenvolupament més fàcil d'una aplicació API.
- Per la part de la IA, hi ha llibreries molt comunes com Keras o Pytorch que permeten desenvolupar en llenguatge Python un model d'IA basat en xarxes neuronals.

El sistema de control per a poder funcionar correctament necessitar tenir una lògica darrere que permeti assegurar en qualsevol cas que l'elecció sigui la correcta. S'han ideat diferents mètodes generals per a poder fer-ho, però el més usat és el control difús o *Dizzy logic*. Aquest, a diferència de la lògica booleana clàssica, es treballa en valors entre 0 i 1 en lloc de valors binaris obert o tancat. En el cas de tenir un sistema on no són necessaris valors mitjans, s'utilitza llavors la lògica booleana clàssica, on hi ha dos estats únics, fals (normalment representat amb un 0) o cert (normalment representat amb un 1), simplificant molt el sistema de control.

Llavors, aquest treball neix de la necessitat d'automatitzar una màquina basada en el RCE. Actualment, aquesta funciona mitjançant un sistema de control bàsic, sense la part intel·ligent. El rendiment de la màquina llavors no s'adapta al que passarà també al futur i per aquest tipus de màquina pot resultar molt pràctic poder conèixer com evolucionarà la màquina en les pròximes hores. És per això que, realitzant una nova lògica de control usant prediccions, es pot intentar millorar el rendiment actual i optimitzar el sistema.

1.5 Enfocament i metodologia

Aquest treball s'enfocarà en el desenvolupament d'un sistema. Es realitzarà en un entorn de desenvolupament i s'exposarà en la memòria el que s'ha dissenyat i el perquè.

Per a fer la programació, es farà servir l'IDE Pycharm, on s'implementaran les tres parts del sistema de control en el llenguatge Python i s'empraran biblioteques com FastAPI, Keras per facilitar-ne el desenvolupament.

Llavors, l'explicació de cada part del sistema de control es dividirà en tres parts: dissenyar les característiques de cada component, aplicar en codi les decisions preses i validar-ho.

Per a fer tot això, s'ha de tenir ordenat en el temps tot el que es realitzarà. Ho podem fer amb una taula, però per fer-ho de forma molt més gràfica, ho farem amb un diagrama de Gantt, que ens permetrà ordenar tot el que farem de forma visual.

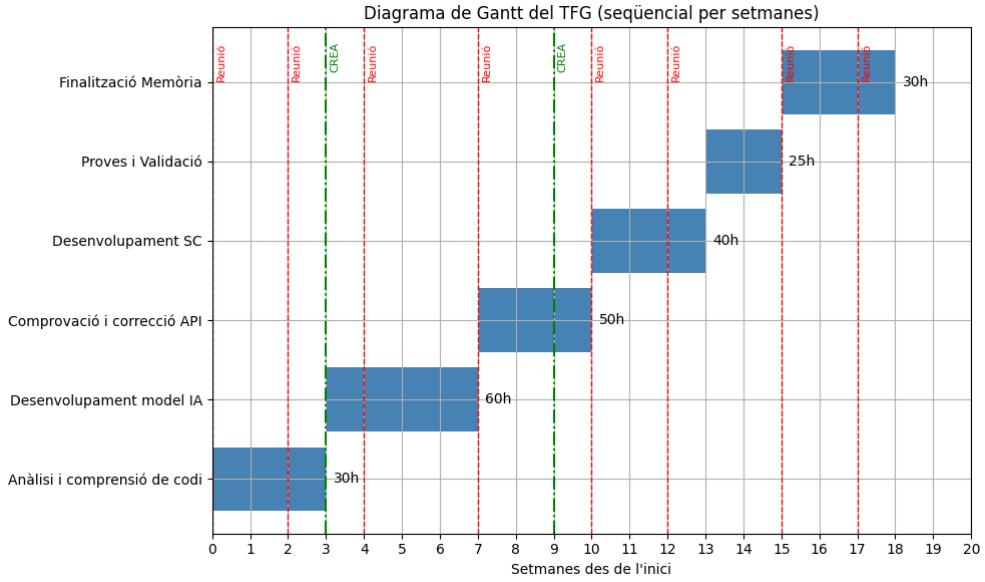

Figure 1: Diagrama de gantt

Primer es va fer una anàlisi del codi de l'API i es va comprovar el seu funcionament. Es va intentar executar-ho, però no es va arribar a fer-ho, cosa que va suposar haver de refer el codi de l'API i del model d'IA. El primer pas va ser desenvolupar el model d'IA, on bona part del temps va ser aprenentatge de programació de xarxes neuronals i comprensió dels millors tipus de models. Un cop desenvolupat un model amb resultats correctes, es va arreglar l'aplicació API perquè funcionés i es van corregir i millorar les prediccions. Finalment, es va realitzar el sistema de control i la integració de l'API a aquest. Un cop tot va estar desenvolupat, es van dur a terme proves per validar el correcte funcionament i es va finalitzar la redacció de la memòria i realització de la presentació.

Però tot això no ho podia fer sol perquè necessitava saber com anava i si era correcte tot el que estava fent. El seguiment de les reunions va variar el que s'estava desenvolupant. Durant

l'aprenentatge del codi, es van fer 2 reunions una al principi i una al cap de dues setmanes per comprovar si estava entenent el funcionament i fer-me preguntes sobre la lògica. Llavors, en la tercera setmana es va fer la primera reunió amb el CREA on em van ensenyar el prototip, com funcionava per dins... Llavors, en el desenvolupament de la IA es van fer dos reunions per saber com era el rendiment d'aquesta i el que s'havia de millorar. Durant el desenvolupament de la API es va fer una reunió amb el CREA del que s'havia desenvolupat sobre la IA i la API, i es plantejar que fer al sistema de control. Llavors es va fer una reunió que englobava el sistema de control i la validació de tot.

Però com que el TFG no només era la part de programació sinó que també de redacció d'una memòria, es va plantejar també com es faria la redacció. Se seguiria una línia similar entre el que es desenvolupava i el que es redactava. En les tres primeres setmanes, mentre analitzava el codi, es va buscar informació rellevant dels models d'IA, els sistemes de control, les API... per desenvolupar una base teòrica que acabaria sent el *Marc Teòric*. Llavors, pel desenvolupament de cada component es va per la programació i de forma paral·lela s'explicava el disseny i la implementació a codi d'aquest. En les proves, el que s'anava testant es documentava també. I un cop finalitzat, es va comprovar que tot el redactat estigués clar i amb tota la informació necessària. Es fa fer el desenvolupament de la presentació per resumir i explicar de forma clara tot el treball.

1.6 Estructura del treball

El treball es dividirà en diferents seccions, on es començarà comentant la teoria darrere de la implementació del sistema de control i el model predictiu, continuant amb la implementació d'aquests explicant el procés que fa el sistema per a seleccionar l'acció a fer, la transmissió de dades en la comunicació... Se seguirà el següent ordre:

- **Marc teòric**: S'explicarà els dos desenvolupaments de forma teòrica, sobre la tecnologia usada, la teoria darrere dels mètodes usats... i el funcionament del sistema de la màquina basat en RCE per a contextualitzar el perquè de les decisions preses al sistema de control.
- **Disseny del sistema de control**: S'explicarà el desenvolupament del disseny del sistema de control i de la IA, sobre les decisions preses i el perquè d'aquestes. També s'explicarà el procés que se seguira al prendre les decisions de forma teòrica.
- **Implementació**: S'exposarà els trets més rellevants en el desenvolupament del programari. S'explicarà en el codi el procés que se seguirà i el seguiment de les dades pels diferents processos.
- **Proves i validació**: S'exposaran els resultats obtinguts pel sistema de control, explicant el sentit que tenen per les diferents proves usades.

- **Conclusions**: S'exposarà les conclusions del treball, indicant les limitacions del programari i les possibles línies de recerca futures.

2 Marc teòric

2.1 Funcionament RCE

El sistema basat en el RCE o Radiative Cooling and Solar Heating és un sistema tèrmic dual automàtic que combina la calefacció solar foto-tèrmica i la refrigeració radiativa passiva. La clau del funcionament resideix en l'ús d'un material termo-responsiu, que es modifica dinàmicament en funció de la temperatura ambiental. A baixes temperatures, el material afavoreix l'absorció de radiació solar, mentre que en temperatures altes, s'augmenta l'emissió de raigs infrarojos, cosa que provoca una refrigeració radiativa dissipant la calor a l'exterior [3].

El sistema RCE és un sistema que permet operar tant en un mode de refrigeració com en un mode de calefacció. Per això, es farà servir una màquina que tindrà dos circuits independents, un per temperatura freda i un per calenta, amb els seus corresponents tancs d'aigua.

Per a guardar l'energia tèrmica de l'aigua, es fa servir l'energia solar. Per a guardar l'energia en els dos modes, farem servir la diferència de temperatura entre el dia i la nit. Durant la nit, com que la temperatura de l'ambient és freda, farem servir aquesta per refredar l'aigua del circuit perquè, quan passi pel tanc fred, aquesta refredi l'aigua del tanc. El mateix passarà durant el dia, però a la inversa, on s'aprofitarà l'escalfament de l'aigua a causa de la temperatura de l'ambient per escalfar l'aigua del tanc.

La part crucial d'aquest sistema és quan canvien el mode de recol·lecció de calor i per a fer això es fan servir dues vàlvules que permeten l'intercanvi dels circuits entre fred i calor. A més a més, hi ha una placa de vidre damunt de la placa solar que recol·lecta la calor.

El funcionament d'aquesta placa de vidre ens permetrà maximitzar la calor que es rebrà al circuit. Quan el vidre està damunt de la placa, es crearà entre el vidre i la placa l'efecte hivernacle, que provocarà que la llum solar no s'escapi i provoqui més temperatura. Com que a la nit volem l'efecte contrari, es treu el vidre perquè la màxima llum possible s'escapi i no escalfi molt la placa.

2.2 Xarxes neuronals i Machine Learning

L'aprenentatge automàtic o *Machine Learning* és una branca de la intel·ligència artificial que està sent un pilar en la tecnologia actual. Aquesta estén la programació clàssica de control per regles, on s'executarà una acció corresponent en relació a certes regles o condicions. Enlloc d'això, el sistema mateix aprendrà a partir d'unes dades els patrons entre les dades per llavors poder predir i decidir accions.

Com que aquesta tecnologia és molt amplia, s'han classificat tres tipus de Machine Learning:

- **Supervisat**: L'algoritme aprèn a partir d'una etiqueta amb la resposta correcta. Per exemple, en una base de dades, hi ha una imatge associada amb el tipus d'animal.

Aquest tipus de ML es fa servir en sectors de detecció de figures en imatges, detecció de fraus, prediccions de demanda...

- **No supervisat**: En aquest, les dades no tenen cap etiqueta amb la resposta. Llavors el sistema mateix ha de descobrir, mitjançant la busqueda de patrons o estructures ocultes, quin és el resultat a predir.

  Aquest tipus de ML s'usa en detecció d'anomalies, compressió de dades, segmentació de mercat...

- **Reforç**: L'algoritme aprèn per assaig i error. Quan una predicció que realitza és vàlida, el sistema rebrà una recompensa, però si és incorrecta, rebrà un càstig. Llavors, aprendrà quina és l'estratègia que retorna els millors resultats.

  Aquest tipus de ML s'usa sobretot en els videojocs, la robòtica, control de sistemes...

A més a més, es poden classificar les xarxes per la tècnica:

- **Xarxes neuronals artificials (ANN)**: Les xarxes neuronals s'inspiren en com funciona el cervell humà. Aquestes, en el model, són unitats de processament interconnectades entre elles i el seu funcionament bàsic és rebre, processar i generar dades. Hi ha tres tipus de neurones, les d'entrada que reben les dades de l'usuari, les ocultes que calculen nous valors de dades i les de sortida que calculen el resultat final i el retornen.

  Aquests tipus de model, abans de fer qualsevol tipus de predicció, necessiten tenir una base de dades d'entrenament, que pot ser un fitxer d'exemples. Durant la fase d'entrenament, el fitxer buscarà entre les dades possibles patrons, regles... i per saber que està entrenant correctament, va comparant l'error entre la sortida real i sortida predita. Per arreglar l'aprenentatge i minimitzar l'error, s'utilitza un procés anomenat retropropagació, on de forma dinàmica, s'autoajusta els pesos de les neurones per minimitzar l'error [4].

  Encara que es busqui minimitzar l'error durant l'entrenament, que la IA finalitzi l'entrenament amb un error molt mínim, això no vol dir que sigui una bona IA, ja que hi ha un concepte, el overfitting, que farà que la IA tingui errors. L'overfitting es dona quan el model s'ajusta massa a les dades d'entrenament. Això significa que ha aprés de les dades patrons falsos. Degut a això, si el model rep dades que no siguin les d'entrenament, serà molt probable que la predicció sigui erronea. Per a solucionar això, hi ha tècniques a desenvolupar el model que evita el overfitting. Aquestes tècniques s'explicaran durant el disseny i implementació del model de la IA [5].

- **Xarxes Neuronals Profundes (Deep Learning)**: Les xarxes neuronals profundes, a diferència de les ANN, tenen una estructura més complexa. Aquestes arriben a tenir més de tres capes, arribant fins als centenars o milers de capes. Amb aquests models, es pot fer

aprenentatge no supervisat i es pot fins i tot avaluar i refinar els resultats per augmentar la precisió.

Recentment, com que per fer deep learning es necessiten moltes unitats de processament, s'està utilitzant molt serveis de computació distribuïda al cloud.[6]

2.3 Llenguatge de Programació

A l'hora de desenvolupar un sistema de control i una API sobre una IA és molt important triar un llenguatge de programació que s'adeqüi als requisits i objectius de la solució. Per tant, per la IA els llenguatges considerats són:

- **Python**: És el llenguatge generalment usat per a desenvolupar models d'IA predictius. Això és gràcies al seu gran ecosistema amb llibreries molt útils com Pandas, Tensorflow, Pytorch... i per tenir una àmplia documentació relacionada.
- **C++**: És un dels llenguatges amb millor rendiment, però amb una complexitat major i poques llibreries relacionades amb IA.
- **Java**: Llenguatge robust, però no tan usat en el sector de la IA i Machine Learning, malgrat tenir llibreries com Deeplearning4j que permet aquest desenvolupament.

Finalment, s'ha triat Python com el llenguatge per fer la IA, en tenir moltes llibreries per desenvolupar IA i ser aquestes molt poderoses.

Amb relació a un sistema de control, els llenguatges han de tenir seguretat del procediment, han de ser flexibles i amb comunicació a altres serveis, els millors llenguatges són:

- **C/C++**: Llenguatges molt utilitzats en sistemes de control, permetent un rendiment òptim.
- **Python**: No arriba a ser tant eficient com C o C++, però és molt bona opció degut a la flexibilitat i gran ecosistema de llibreries per la comunicació que té.
- **Java**: Opció robusta pel seu suport a serveis distribuïts, però no és del tot bona opció a causa de la sobrecàrrega que arriba a tenir en l'entorn d'execució.

S'ha seleccionat realitzar el sistema de control en Python, ja que fent ús de les llibreries i interfícies que té, facilita molt una bona i eficient comunicació entre l'API i sistema de control.

2.4 Comunicació basada en sol·licituds

La comunicació basada en sol·licituds és un model de comunicació entre diferents sistemes en què un component fa una petició i un altre respon. De forma més col·loquial, és com una conversa entre els components, on un parla i l'altre el contesta.

Aquest tipus de comunicació té les següents característiques:

- **Esperar una acció**: El component que espera no farà res fins que algú el cridi.
- **Sincronisme/Asincronisme**: El component que fa la petició pot o bé esperar la resposta (Síncrona) o bé continuar el procés i ja arribarà la resposta quan acabi (Asíncrona). En el cas del projecte, les crides a les API son síncrones. Això facilita la comunicació, però si s'amplia l'API, l'espera resultarà cada cop més gran, però com que no treballa a temps real el sistema de control no és tan desavantatjós.
- **Codi de resposta**: Es fan servir codis universals per a veure com ha anat el procés del component que respon. Per exemple, 200 significa que tot ha anat bé, 404 que no s'ha trobat el que s'ha buscat, 500 significa error del servidor...

La comunicació més comuna és mitjançant HTTP. Aquest és un protocol de transferència de dades en serveis web, però es pot usar en altres sectors com en API RESTful. Aquest tipus de comunicació té les següents característiques:

- **Operacions bàsiques**: Aquesta comunicació es basa en operacions HTTP, que cadascuna fa un procés diferent. Els més usuals són el GET per a agafar dades, el POST per transformar unes dades que es passen en altres, el PUT per a actualitzar una dada i el DELETE per a eliminar una dada.
- **Fitxer dades**: Els fitxers de dades poden ser de dos tipus. Hi ha el fitxer JSON, que és fàcil de llegir i agrupa les dades amb unes claus úniques. A més a més, hi ha el fitxer XML, on els valors estan associats amb unes etiquetes, afavorint una descripció detallada.

2.5 Aplicació API

Una API o interfície de programació d'aplicacions és un conjunt de protocols que permeten la comunicació entre diferents programes [7].

Aquesta pot exposar funcions, dades... perquè altres components hi puguin accedir de forma externa. Com que amb això estem separant la lògica en diferents components separats, pot resultar realment útil per a dissenyar sistemes modulars i escalables.

Les API poden ser de diferents tipus:

- **RESTful**: Estil basat en en el protocol HTTP. És senzill i escalable [8].
- **SOAP**: Protocol estricte basant en XML. Ofereix característiques avançades de seguretat [8].
- **Websockets**: Objectes que permeten la comunicació bidireccional en temps real entre client i servidor. Manté una connexió oberta de forma constant.[9]

- **GraphQL**: Llenguatge de consulta desenvolupat per Facebook que permet sol·licitar exactament les dades que es necessiten [9].

Llavors, es poden desenvolupar les API en relació a si es vol publicar o no.

- **API pública**: Qualsevol pot utilitzar-la de forma lliure. Per controlar l'accés, s'utilitzen claus d'accés per evitar atacs.
- **API privada**: Només pot ser usada per l'organisme que l'ha desenvolupada. Això assegura el control i la seguretat.

En el nostre projecte hi haurà els dos tipus d'API. La API que s'ha desenvolupat que fa les prediccions serà privada, ja que només serà d'utilitat per la màquina objectiu. Però per la predicció de condicions meteorològiques aquesta serà una publica, ja que existeixen API a la xarxa que ja fan el càlcul de les condicions meteorològiques

2.6 Sistema de control

Un sistema de control és un conjunt d'elements interconnectats que tenen com a finalitat guiar i dirigir un sistema. Aquests poden ser sistemes manuals o automàtics.

Es pot classificar els sistemes de control en 2 tipus:

- **Sistemes de control obert**: Aquest tipus de sistemes no tenen informació de com funciona la màquina que es vol controlar [10]. Això vol dir que si no funciona correctament, el sistema no s'adaptarà a les condicions actuals.

  Hi ha diferents exemples, com un semàfor que cada llum està basada en un temporitzador, un forn que s'escalfa el temps indicat sense tenir en compte el de dins... [10]

- **Sistemes de control tancat**: Aquest tipus és el contrari als de control obert. Mitjançant sensors o algun tipus de variable, es va controlant la sortida de la màquina a controlar. Gràcies a això, es poden ajustar les decisions de forma automàtica [10].

  Aquest tipus de sistemes són usats en electrodomèstics automàtics, com un termòstat, l'aire condicionat... [10]

Finalment, hi ha diferents tipus de sistemes de control d'acord amb com estan construïts. Com que en el nostre cas és mitjançant codi, el nostre sistema de control serà l'únic tipus de software, el **control per computadora**.

3 Disseny del sistema de control

3.1 Requeriments del sistema

Abans d'explicar com es desenvoluparan la API i el sistema de control, s'han d'especificar els diferents requeriments que han de complir per a, llavors, dissenyar la lògica en base a aquests. Hi haurà quatre tipus de requeriments.

3.1.1 Requeriments funcionals

Les funcionalitats que el sistema de control i la API integrada amb IA han de complir les següents funcionalitats:

- El sistema ha de decidir amb prediccions a futur.
- La API ha de retornar prediccions a futur en relació a la temperatura, temps atmosfèric, producció de la màquina i demanda.
- El sistema ha de rebre dades a temps real.
- El sistema ha de registrar les decisions que s'ha pres.

3.1.2 Requeriments no funcionals

Les funcionalitats generals del sistema són:

- **Escalabilitat**: Adaptació fàcil si s'afegeixen més sensors o dades.
- **Fiabilitat**: Sistema robust als errors.
- **Eficiència**: La predicció de dades s'ha de fer en un temps raonable.
- **Portabilitat**: S'ha de poder executar el sistema independentment de l'entorn.

3.1.3 Requeriments de software

Finalment, s'especificarà el software necessari per a la correcta execució del sistema:

- **Sistema Operatiu**: Es pot executar tant en Windows com Linux.
- **Entorn d'execució**: S'executarà el sistema amb Docker per a assegurar la portabilitat.
- **Llenguatge**: S'utilitza Python 3.12 pel desenvolupament de la API i del sistema de control.
- **Sistema de control**: S'utilitza Git per al seguiment del codi.

- **Llibreries principals de Python**:
  - *Keras*: Llibreria per al desenvolupament d'un model de IA
  - *Pandas, Numpy*: Llibreries per al tractament de les dades.
  - *Requests, FastAPI*: Llibreries per al desenvolupament d'una aplicació API i fer crides a aquesta aplicació.

3.2 Disseny de la IA

3.2.1 Elecció del model

A l'hora de desenvolupar un model de IA, és molt important tenir clar quin model triar, ja que n'hi ha molts tipus, de models. En el cas de fer prediccions, es triaran les xarxes neuronals.

Dintre de les xarxes neuronals, hi ha molts tipus de xarxes.

- **Xarxa Neuronal Feedforward**: Aquesta xarxa permet el flux de dades en una direcció, permetent una classificació bàsica.
- **Xarxa Neuronal Recurrent**: Tenen connexions per recordar informació temporal, permetent fer ús de sèries temporals.
- **Xarxes Convolucionals**: Poden captar patrons especials, com formes o textures, usats generalment per classificació i detecció d'objectes.
- **Xarxes Generatives Adversàries**: Poden generar imatges, augmentar les dades creant dades sintètiques...

D'aquestes, la única que ens servirà serà la Xarxa Neuronal Recurrent, ja que les nostres dades són seqüències temporals. Aquesta estarà dividida en 2 tipus, les LSTM i les GRU. El funcionament és el mateix, però les LSTM són més complexes però més lentes i amb una gran capacitat de memòria. Les GRU són menys complexes i més ràpides, però amb menor capacitat de memòria. Com que el nombre de dades per predir pot variar i, tenir seqüències grans, es faran servir les LSTM.

3.2.2 Preprocessament de les dades

Un pas important pel correcte aprenentatge del model és el preprocessat de les dades. Els models basats en aprenentatge automàtic són molt sensibles a l'escala i la qualitat de les dades.

Unes dades no normalitzades poden provocar que unes característiques tinguin més importància que unes altres. Per a solucionar això, transformarem les dades en un rang finit per a què totes tinguin una importància igual.

Un conjunt de dades de qualitat garanteix que no hi hagi soroll en les dades, és a dir, que no hi hagi valors que facin que el sistema aprengui regles falses. També s'ha d'evitar els *outliners*, valors que són extrems en relació al conjunt de dades, ja que pot provocar que el model aprengui valors incorrectes.

Llavors, al moment de rebre les dades, transformarem els valors amb un escalador en valors en el rang finit i eliminarem tot tipus de valors que puguin empitjorar l'aprenentatge.

3.3 Integració amb el sistema de control

La integració del model de IA amb el sistema de control es farà mitjançant una aplicació API. Per a fer això, aquesta API es farà amb la llibreria FastAPI, que proporciona una interfície eficient i fàcil de fer servir. Per a explicar de forma teòrica com desenvolupar l'aplicació, es dividirà l'explicació en les parts més importants.

3.3.1 Diagrama de seqüència

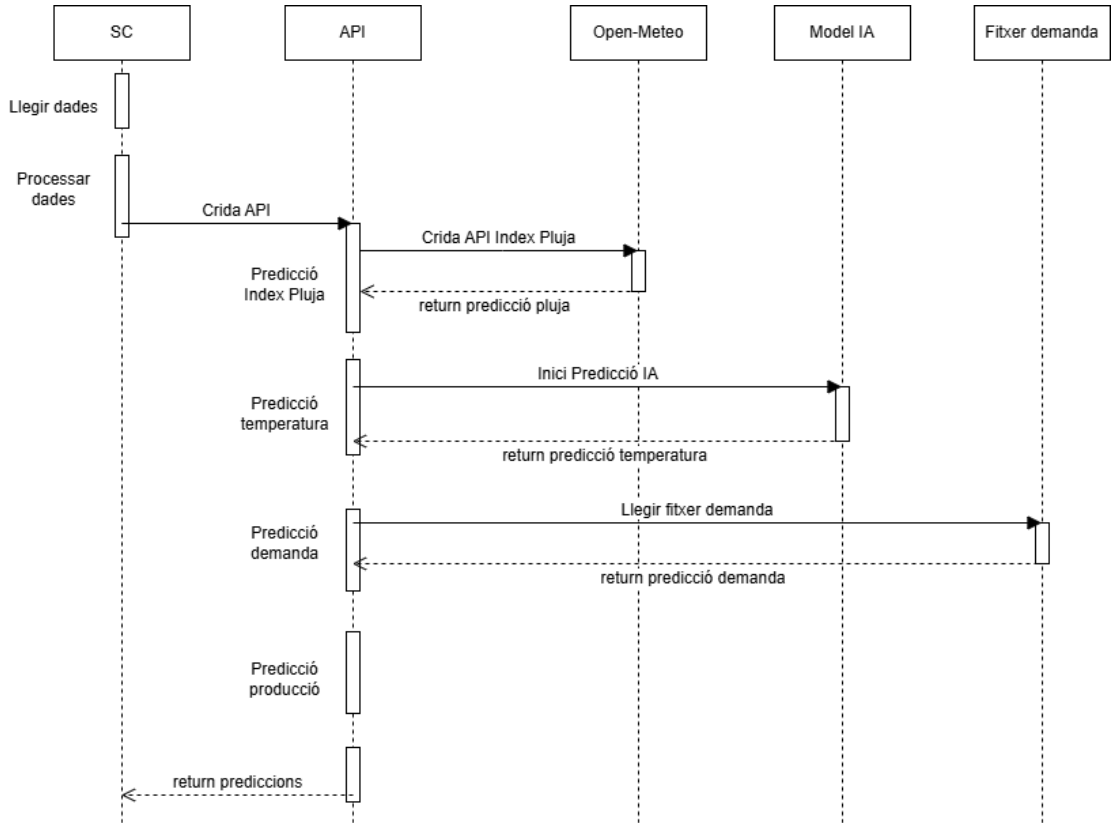

Figure 2: Diagrama de seqüència UML de la API

Ara explicant-ho de forma més general, la API tindrà 5 processos diferents. Un cop el sistema de control envii les dades a la API, aquesta iniciarà el procés de predicció.

Començarà amb la predicció de l'index de pluja, on es podrien fer 2 diferents tipus de desenvolupaments. D'una banda, es podria desenvolupar un model de IA propi, que rebi dades temporals i retorni l'index de pluja. L'altre és utilitzar una API online gratuïta que pugui fer aquesta mateixa predicció. Com que no tenim a la disposició un fitxer clar de dades per utilitzar, la primera pot ser difícil de desenvolupar amb predicció. Llavors, la segona és molt bona opció. Però, no es pot fer servir qualsevol. S'ha de trobar una que considerem que doni bons resultats. En el treball s'han provat 3 tipus d'API.

- **AEMET**: Va ser la API que estava al desenvolupament inicial. Els valors que retornava no eren molt versemblants. A més a més, era molt comú que et retornés un error de connexió.
- **Api de la NASA**: API que retorna una bona predicció generalment, però pot arribar a ser força lenta de respondre.
- **Meteo-Sat**: API que retorna una predicció correcta, en un temps curt.

S'ha seleccionat finalment Meteo-Sat degut a que les prediccions eren força acurades i la resposta era curta.

Per calcular la temperatura, farem servir aquí sí una IA pròpia. Per a fer això, amb les dades que s'han passat del Sistema de Control, es farà la predicció.

Per calcular la demanda, es llegirà un històric de dades de tot un any i es retornaran aquelles prediccions que ens interessin.

Finalment, per predir la producció farem servir una predicció analítica, o sigui, farem servir una formula per calcular la potència teòrica.

Un cop tenim totes les prediccions, les ajuntarem en un sol fitxer JSON i el retornarem al Sistema de Control.

3.3.2 Tipus de API

Abans de definir com serà per dins la lògica de la API, s'ha de decidir de quin tipus serà. Hi ha dos tipus:

- **REST**: Més que un protocol, és una arquitectura per a serveis web. S'utilitza per operacions de HTTP i es pot usar en HTTP i URL. El fitxer general per a comunicacions sol ser el JSON però es pot usar el XML. A diferència de SOAP, no fa falta cap contracte predefinit com WSDL en SOAP [11].

- **SOAP**: Es un protocol estàndard on dues aplicacions es comuniquen amb missatges XML. Aquest funciona sobre protocols HTTP i té suport per seguretat avançada. Malgrat els avantatges, la comunicació té una estructura força rígida, ja que s'ha de seguir un format concret [12].

De forma general, es fa servir l'arquitectura REST en les API, al ser més senzill i lleuger que SOAP, a l'utilitzar directament HTTP, fitxers JSON que són fàcils de tractar i no necessita definicions complexes com el WSDL.

3.3.3 Recepció de les dades

El sistema de control tindrà les dades necessàries per a fer les prediccions. Aquest, en el moment de buscar prediccions a futur, farà un crida a la API. Per a fer la crida, com s'ha dit prèviament, es farà mitjançant una crida REST. S'ha de declarar com es farà la corresponent crida:

- **Operació REST**: Hi ha quatre operacions bàsiques, però de les quatre n'hi ha dues que potser ens pot servir, GET o POST, ja que el Sistema de Control agafarà dades. Malgrat això, com que GET serveix per a agafar dades de la API i POST serveix per aconseguir dades de la API mitjançant un procés, és clar que la millor opció és POST.

- Dades: Les dades s'enviaran pel body de la request. En aquest, s'enviara el fitxer en el format JSON. Aquest tindrà la següent forma, ja que contindrà les dades necessàries per a fer les prediccions.

```
data:[
    {
        "cold": 14.49,                  # Temperatura del tanc fred
        "hot": 20.3,                    # Temperatura del tanc calent
        "wind_vel_m_s": 0.863,          # Velocitat del vent en m/s
        "solar_rad_w_m2": 0.0,          # Radiació solar en w/m^2
        "ir_rad_w_m2": 327.848,         # Radiació infrarroja en w/m^2
        "day_sin": 0.9469301294951056,  # Moment del dia en format sin
        "day_cos": 0.3214394653031617,  # Moment del dia en format cos
        "year_sin": 0.8263541987239096, # Dia de l'any en format sin
        "year_cos": -0.5631507242749186 # Dia de l'any en format cos
    },
    {
        ...
    }
]
```

Aquest JSON contindrà, en el camp "data", 24 instants de temps, on un instant conté les 9 dades que es mostren prèviament al JSON. Les columnes són els valors representats en un temps concret.

Com es pot veure, no es mostra enlloc el moment del dia en format *date*, ja que aquesta s'ha transformat en quatre columnes, el sin i cos del dia i el sin i cos de l'any. Això és fa per a què el model de IA tingui més facilitats a l'hora d'entrenar i predir. La raó és que, a l'hora de buscar patrons durant l'entrenament, es busca que hi hagi relacions i, si busca patrons en relació al dia, acabarà relacionant que del dia 1 de gener i 2 de gener que hi ha un dia, però si busca entre 31 de desembre i 1 de gener, ho intentarà entendre com si la diferència fos d'un any. Per això, si ho transformem en funcions sinus i cosinus, tindrem que les dates seran en un format completament cíclic. És el mateix funcionament pel dia, ja que entre les 23:59 i 00:00, hi ha 1 minut i el model de IA pot entendre com si hi hagués 23:59 hores de diferència.

3.3.4 Predicció pluja

A l'hora de predir la pluja, com s'ha dit prèviament, farem servir una API externa, la qual és Meteo-Sat. Però per a fer-la servir, com que no és nostra, s'ha de comprovar que és el que demana perquè ens retorni la predicció vàlida. En aquesta API, és necessari indicar la posició de la qual volem aconseguir la predicció i la zona horària de la qual es fa la crida.

Per la posició, serà clau declarar els valors d'aquesta amb la màxima precisió possible per a poder marcar la localització que vol. Posaré un exemple de bona i mala declaració per fer-ho més entenedor:

- **Bona declaració**: Lat = 41.608129, Long = 0.623524. Posició de l'edifici Escola Politècnica Superior.
- **Mala declaració**: Lat = 41.60, Long = 0.62. La distancia entre aquest punt i l'escola politècnica superior supera el mig kilòmetre.

La declaració de la zona horària de la qual volem buscar les prediccions també és molt important. Les dades, usualment, es tracten en zona horària global, la UTC. Aquesta serveix per a poder tenir un correcte tractament de les dades a qualsevol part del món. Malgrat això, no tot el món està en format UTC. Per a solucionar això, indicant la zona horària podrem assegurar que sempre ens retornarà les dades del dia d'avui. Si no fos així, podria passar que si es fes la crida en una hora crítica, com les 12, ens retornés les dades del dia d'ahir, ja que a Espanya estem en UTC+02:00, o sigui, 2 hores avançats de UTC, cosa que per l'API externa donaria com a resultat que es buscaria al dia previ.

Llavors, l'API ens retorna les dades del dia complet. Per tant, hem d'agafar només aquelles que siguin del nostre interès. En el nostre cas, ens serà interessant agafar les dades atmosfèriques relacionades amb l'índex de pluja, la velocitat del vent, la radiació solar i la radiació infraroja. Com que ens serà obligatori adquirir les tres últimes condicions durant les prediccions de temperatura, aconseguirem durant la predicció de pluja únicament l'índex de pluja.

3.3.5 Predicció temperatura

Per a predir la temperatura es farà servir el model d'IA que s'ha desenvolupat. Per a fer la predicció, s'ha de marcar quins seran els paràmetres d'entrada.

- **Entrada**: Perquè el model pugui predir, farem servir instàncies de temps. Aquestes instàncies seran enviades des del Sistema de Control fins a l'API. A l'hora de saber quanta informació ens ha d'aportar el Sistema de Control, hem de declarar correctament una finestra temporal.
- **Finestra temporal**: A l'hora de predir en models LSTM, hem de declarar quina volem que sigui la nostra finestra temporal. Podem pensar diferents rangs d'hores, però s'ha comprovat que un rang de 6 hores és l'òptim. Un rang de dades baix, de 2 hores per exemple, pot provocar que el model d'IA no arribi a trobar cap patró en aquestes dues hores de dades, cosa que provocaria una incorrecta predicció. Però si volem usar un rang gran, com de 12h o 24h, ens pot passar el contrari, tenir tantes dades que provoqui soroll i les prediccions tinguin errors. Llavors, el millor és un punt mitjà i 6 hores és una molt bona finestra.
- Sortida: La sortida serà una llista temporal de les prediccions en instants de temps de forma seqüencial, és a dir, primer la predicció en el moment t0+15 min, t0+30 min... fins a completar les 2 hores.

3.3.6 Predicció demanda

La predicció de la demanda no serà una predicció com a tal. En lloc de tenir un model de predicció que, amb un conjunt de dades, ens permeti arribar a predir la demanda futura, es tindrà un històric de dades amb les demandes de tot un any. Això tindrà beneficis, com que la lògica del codi serà més ràpida i el cost computacional serà el d'agafar les dades del fitxer, però tindrà greus inconvenients, com que se suposa que la demanda de l'any serà cada any la mateixa i no es tindrà en compte cap variació.

El fitxer on està l'històric de demanda haurà de tenir el següent format, en un fitxer .csv.

- **Date/Time**: Columna on hi haurà els valors de dates en format dia, hora i minut de l'any
- **Heating [J]**: Columna on hi haurà els valors de la temperatura del tanc calent en el minut "t".
- **Cooling [J]**: Columna on hi haurà els valors de la temperatura del tanc fred en el minut "t".

Per a fer la predicció, s'haurà d'indicar l'històric de dades i un rang de dates que serà el rang on s'agafaran les prediccions. Llavors, és llegirà el fitxer corresponent i se seleccionaran les demandes dins del rang.

3.3.7 Predicció producció

Per a fer la predicció de la producció del sistema RCE, farem servir una predicció analítica, és a dir, farem servir una fórmula matemàtica que permetrà aproximar la producció real. La fórmula corresponent és la següent:

$$\text{PROD}_{tn} = \text{mode} \cdot \rho \cdot c_p \cdot (T_{\text{pred}} - T_0)$$

Cada variable significa:

- $\textbf{PROD}_{tn}$: Producció d'energia en l'instant t_n [J].
- **mode**: Factor de mode, que val 0.15 para calefacció o 0.05 per a refrigeració.
- ρ: Densitat del fluid [kg/m^3].
- c_p: Calor específic a pressió constant del fluid [J/(kg·K)].
- $T_{\textbf{pred}}$: Temperatura predita del fluid [°C o K].
- T_0: Temperatura inicial del fluid en el instant t_0 [°C o K].

Aquesta fòrmula, heretada de l'antic projecte del Christian, calcula l'energia tèrmica que pot produir el sistema d'intercanvi de calor en un instant t_n.

3.3.8 Ajuntar les dades

Finalment, un cop s'han aconseguit les quatre prediccions, s'ha de retornar les dades al sistema de control. Per a fer això, es crearà un fitxer JSON amb les prediccions. S'aniran afegint a un mateix fitxer les quatre prediccions que s'han aconseguit i es retornarà aquest fitxer al sistema de control.

3.4 Disseny del sistema de control

3.4.1 Diagrama de l'arquitectura del SC

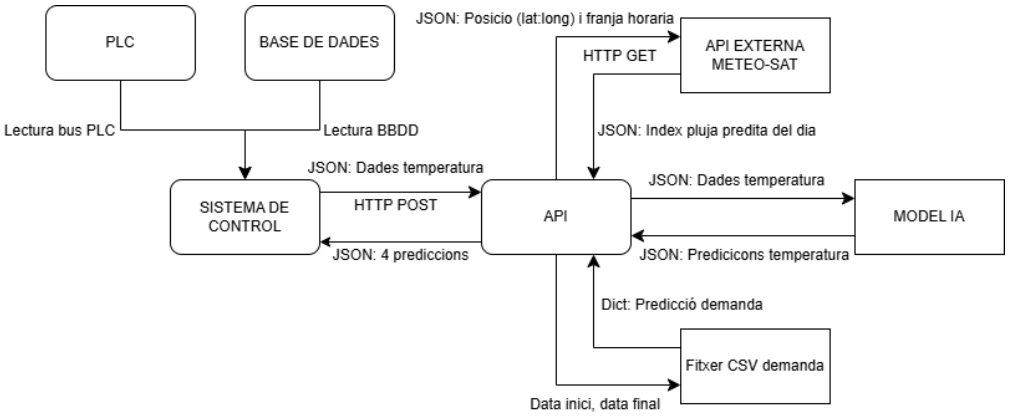

Figure 3: Diagrama de components

3.4.2 Lògica Sistema de Control

El sistema de control serà l'encarregat de dirigir a la màquina. Per a fer això, tindrà dues variables per controlar l'estat, una per saber si està oberta o tancada i una altra per saber el mode. Per determinar com ha d'estar la màquina, durem a terme proves de control cada 5 minuts. En aquesta es comprovarà si en aquell moment es vol produir o no. Per saber si en quins moments s'ha de produir o no, veurem amb les potències predites quines són les que la potència és major i l'energia menor. De forma gràfica es pot representar això amb un exemple reduït, on la demanda d'aquest és de 6500 J.

Temps	T1	T2	T3	T4
P_w	1000	3000	3500	3000
E_bomba	800	1250	1350	1200
Selecció			X	X

Figure 4: Exemple selecció finestra

En aquest exemple se seleccionen les finestres T3 i T4 en tenir una potència major i una despesa d'energia menor.

Llavors, si en el moment actual estem dins d'una de les finestres, comprovarem si realment ens val la pena. Per a fer-ho, calcularem l'índex de rendiment o COP, que es calcula dividint la potència generada entre l'energia consumida pel temps. Si el COP és positiu, significarà que és rendible produir i obrirem la màquina. Si no, assumirem que no podem i tancarem la màquina.

Però, quan es calculen les finestres? Hi haurà un moment que es calcularan per a la temperatura calenta i un per a la freda. Per a la calenta, a la sortida del sol calcularem la demanda de la tarda del mateix dia i de la matinada de l'endemà, que tot això sumarà la demanda total. Llavors amb aquesta buscarem aquelles finestres de la mateixa forma que a la figura 4. Per la temperatura freda la lògica és la mateixa, però es calcularà a la nit i es buscarà la demanda de fred de tot l'endemà.

3.4.3 Diagrama de seqüència

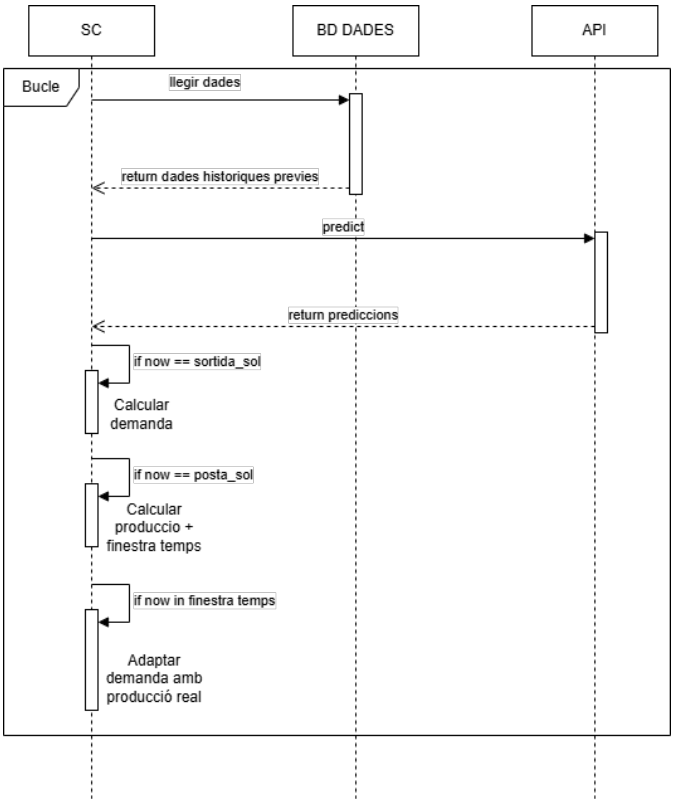

Figure 5: Diagrama seqüència Sistema de Control

El sistema estarà en bucle de forma indefinida. Llavors, en cada iteració llegirem les dades de la temperatura de la base de dades corresponent. Un cop les tenim, farem una crida a l'API i, de forma síncrona, s'esperarà a una resposta. Quan el sistema ja té les prediccions, tindrem tres possibles casos. Primer comprovarem si el temps actual és la sortida o posta del sol. Es fa primer aquestes comprovacions perquè si fos en el cas que sí, el sistema pugui adaptar-se directament des del moment que es troba. Si és el cas que estem dins d'una finestra temporal, farem la comprovació amb la potència real. Si no, tancarem la màquina en el moment actual.

3.4.4 Obrir/Tancar màquina

Per a triar quan s'ha d'obrir o tancar la màquina, hem de decidir en quin moment del dia la producció, tant sigui en fred o calent, permeti arribar a cobrir la demanda. Per a fer això, serà necessari saber quina demanda arribarà a tenir el sistema i farem una diferenciació entre fred i calent.

- **Calor**: Per a saber quant hem d'arribar a produir, necessitarem saber la demanda de calor de la tarda del dia 1, és a dir, el dia que es fa l'execució i el matí del dia 2. Es calcularà el dia 1 a la sortida del sol.
- **Fred**: Necessitarem saber la producció del dia 1 per a cobrir la demanda del dia 2. Es calcularà el dia 1 a la posta del sol.

Llavors, un cop calculada la demanda, necessitarem trobar en quins moments la producció és major amb una despesa energètica mínima. Per a fer això, trobarem quines potències predites tenen un valor més alt i una despesa energètica més baixa i guardarem en una llista de finestres la finestra en la qual es produeix la corresponent part de potència total necessària.

En el cas que el temps t no estigui en una franja dins de la finestra de control, la màquina romandrà tancada, ja que s'ha decidit que no era un bon moment. En el cas que sí que estigui dins de la finestra, es calcularà el COP, el coeficient de rendiment. Aquest té la següent forma:

$$\text{COP} = \frac{Prod\_tn}{E\_Bomba} = \frac{Prod\_tn}{P\_bomba * Time\_step}$$

Si aquest valor és superior o igual a 1, significarà que el rendiment en el temps t permet aconseguir més potencia de la que es consumeix i obrirem la màquina. Si és inferior a 1, voldrà dir que la màquina no és capaç de produir suficient. Llavors, s'acceptarà no cobrir la demanda en aquest temps i es tancarà.

3.4.5 Acció en un time_step

En cada *time_step*, s'ha de comprovar que es produeixi el que realment s'ha predit. Per a fer això, comprovarem la producció real i predita. En el cas que les produccions siguin les mateixes, no farem cap procés addicional, però en el cas que hi hagi alguna discordança, restarem a la demanda la potència real i realitzarem un altre cop la cerca d'una finestra de temps on la producció sigui vàlida. En el cas que s'hagi complert la demanda, s'eliminaran les finestres de control futures.

El temps d'espera entre una predicció i una altra serà de 5 minuts. Aquest temps permetrà un error molt baix en prediccions amb gran diferència de valors.

4 Implementació del sistema de control

4.1 Desenvolupament del model d'IA

4.1.1 Lectura de les dades d'entrenament

Per a poder desenvolupar un model, serà necessari tenir uns fitxers amb dades històriques per a entrenar. En el cas del model per a predir les temperatures, tenim un fitxer amb un històric de dades anomenat "*cleansed.csv*".

Aquest fitxer té 550773 files de dades i, d'aquestes, no hi ha cap fila amb valors buits. Per tant, no farà falta tractament de valors *null*.

Llavors, per llegir el fitxer al codi farem servir la llibreria *pandas*. Aquesta llibreria aporta a *python* una forma de manejar les estructures de dades de forma fàcil i flexible. Hi ha dos tipus d'estructures de dades, les Sèries (Estructura d'una dimensió) i els Dataframes (Estructura de dues dimensions). Com que treballem amb dades historiques on, a cada temps hi ha més d'una dada, l'estructura usada serà els Dataframes [13].

En codi, això es representarà així:

```
df = pd.read_csv("cleansed.csv")
```

4.1.2 Preprocessament de les dades

Com s'ha comentat prèviament al disseny, abans de passar les dades al model, hi ha d'haver un preprocessament de les dades. Per a fer això, hem de saber quines dades volem passar-li al model i quines no i fer un escalat de les dades per a que totes tinguin la mateixa rellevància.

En el primer cas, tindrem la funció process_data que serà l'encarregada de transformar el Dataframe de dades en el format vàlid. Per a fer això, transformarem els temps en 4 variables.

- **day_sin**: Moment del dia en format sin
- **day_cos**: Moment del dia en format cos
- **year_sin**: Dia de l'any en format sin
- **year_cos**: Dia de l'any en format cos

Això té una raó, ja que si no, podríem confondre al model. Posant com a exemple els dies 1 de gener i 31 de desembre, pel model semblaria que hi ha 365 dies de diferència, però, realment, només n'hi ha 1 ja que és cíclic. El mateix passa amb les hores, ja que les 00:00h i les 23:59 només es porten 1 min, però pot semblar que hi hagi hores de diferència. Fent els sinus i cosinus de les funcions farem que sigui cíclic, facilitant al model que aprengui basant-se amb el temps.

Eliminarem totes les columnes que no tinguin tanta rellevància pel model per calcular la temperatura i renombrarem les columnes per a què tinguin un format de text més entenedor per a l'usuari.

```
def drop_columns(df: pd.DataFrame) -> pd.DataFrame:
    """
    Function that drops the columns that are not important for the system
    :param df: The dataframe with the data to be dropped
    :return: The dataframe without the columns dropped
    """
    # Drop columns related to time and specific sensor readings
    df = df.drop(columns=['day_of_year', 'datetime', 'Amb_temp_C', 'P_W'])
    # Drop the first column (likely an index column if not explicitly named)
    df = df.drop(df.columns[0], axis=1)
    # Drop more specific time and sensor columns
    df = df.drop(columns=['Sunrise_min', 'Sunset_min', 'Flow_rate_kg_h',
    'In_RCE_temp_C', 'Out_RCE_temp_C'])
    return df

def process_data(df: pd.DataFrame) -> pd.DataFrame:
    """
    Function that transforms the raw dataframe to the valid one
    :param df: Raw dataframe that was read
    :return: Fixed dataframe with the processed values
    """
    # Transform the datatime column to a datatime format
    df['datetime'] = pd.to_datetime(df['datetime'])

    # Get the time of the row to calculate the sin and cos of the day
    seconds_in_day = 24 * 60 * 60
    time_seconds = df['datetime'].dt.hour * 3600 + df['datetime'].dt.minute
        * 60 + df['datetime'].dt.second
    df['day_sin'] = np.sin(2 * np.pi * time_seconds / seconds_in_day)
    df['day_cos'] = np.cos(2 * np.pi * time_seconds / seconds_in_day)

    # Get the day of the year to calculate the sin and cos
    df['day_of_year'] = df['datetime'].dt.dayofyear
    df['year_sin'] = np.sin(2 * np.pi * df['day_of_year'] / 365)
    df['year_cos'] = np.cos(2 * np.pi * df['day_of_year'] / 365)
```

```
    # Rename name columns
    df = df.rename(columns={
        'Hot_tank_temp_C': 'hot',
        'Cold_tank_temp_C': 'cold',
        'Wind_vel_m_s': 'wind_vel_m_s',
        'Solar_rad_W_m2': 'solar_rad_w_m2',
        'IR_rad_W_m2': 'ir_rad_w_m2'
    })

    # Function to drop the unnecessary columns
    df = drop_columns(df)
    return df
```

4.1.3 Construcció del model

A l'hora de construir el model, és important, a part del model que hem triat per a fer les prediccions, el número de capes i de què està composta cada capa. Com s'ha dit al disseny, s'ha seleccionat un model bidireccional LSTM.

Llavors, la construcció pas a pas del model és:

```
def build_model(input_shape) -> Sequential:
    """
    Function that creates a keras Sequential model and compiles it
    :param input_shape: The size of the input vector
    :return: The raw model that will be trained
    """
    model = Sequential([
        # Capa 1
        Input(shape=input_shape),

        # Capa 2
        Bidirectional(LSTM(128, return_sequences=True,
            activation='tanh', kernel_regularizer=l2(1e-4))),

        # Capa 3
        BatchNormalization(),

        # Capa 4
        Dropout(0.3),
```

```
        # Capa 5
        Bidirectional(LSTM(64, activation='tanh', kernel_regularizer=l2(1e-4))),

        # Capa 6
        BatchNormalization(),

        # Capa 7
        Dropout(0.2),

        # Capa 8
        Dense(2)
    ])
    model.compile(optimizer='adam', loss=Huber(), metrics=['mae', 'mse'])
    return model
```

Com es pot comprovar, el model té 8 capes en total:

- Capa 1:

  Defineix la forma d'entrada del model.

- Capa 2:

  Capa LSTM bidireccional que calcularà 128 unitats per direcció, per tant, 256 sortides per cada pas temporal. Com que tenim una altra capa LSTM posterior, serà important retornar la seqüència completa a la sortida.

  Finalment, aplicarem un regulador kernel, en especific el *l2*, que ajuda a evitar l'*overfitting*, penalitzant pesos molt grans.

- Capa 3:

  Normalitza les sortides de la LSTM bidireccional anterior per accelerar el procès i millorar l'estabilitat del model.

- Capa 4:

  El Dropout serveix per reduir el *overfitting*, ja que, de forma aleatòria, eliminarà neurones de la capa anterior en cada batch durant l'entrenament. La probabilitat d'eliminació s'especifica en valors entre 0 i 1.

- Capa 5:

  Té la mateixa funcionalitat que la capa 2, però amb 128 unitats totals.

- Capa 6:

  Té la mateixa funcionalitat que la capa 3.

- Capa 7:

  Té la mateixa funcionalitat que la capa 4.

- Capa de sortida:

  És la capa final que retorna les dues temperatures predites de fred i calent. S'ha d'indicar que la mida de la capa ha de ser la mateixa que les variables de sortida a predir.

Finalment, s'ha de compilar el model per a preparar-lo per a entrenar. Per a compilar-lo, se li ha d'indicar una funció de pèrdua, un optimitzador i les mètriques que es vulguin observar.

```
model.compile(optimizer='adam', loss=Huber(), metrics=['mae', 'mse'])
```

L'optimitzador adam és un optimitzador que ajusta el ritme d'aprenentatge per paràmetre, fent que sigui robust. La funció de pèrdua Huber és una combinació entre el MSE (Error quadràtic mitjà) i el MAE (Error absolut mitjà), on aquest serà més robust als outliers que el MSE, però més suau que el MAE. Finalment, per comprovar que el model s'entrena correctament, s'ha indicat que a cada iteració es mostri el MAE i MSE per analitzar com va progressant l'entrenament.

4.1.4 Entrenament del model

Finalment, s'ha d'entrenar el model per a que aprengui els diferents patrons. Per a fer això, farem servir la funció train_model:

```
def train_model(model: Sequential, x_train, x_test, y_train, y_test) -> object:
    reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5,
        patience=3, min_lr=1e-6)

    history = model.fit(
        x_train,
        y_train,
        epochs=50,
        batch_size=64,
        validation_data=(x_test, y_test),
        callbacks=[reduce_lr],
        verbose=1
    )
    return history
```

Primer aplicarem un callback (una funció que s'aplica automàticament durant l'entrenament). Aquest és el *ReduceLROnPlateau*, el qual s'activa quan el model s'estanca i no millora el rendiment marcat per la mètrica "monitor", en el nostre cas, el valor de pèrdua "val_loss". A l'aplicar-se el callback, es redueix la tassa d'aprenentatge, ja que és possible que la tassa d'aprenentatge sigui massa alta i se salti els pesos on sigui el model més òptim. De forma col·loquial, seria arribar a la solució fent passes cada cop més petites si s'escau [14].

Finalment, aplicarem la funció *fit* per a entrenar el model. Aquest té els següents paràmetres:

- **x_train**: Llista de valors que serveix per aprendre la temperatura futura.
- **y_train**: Llista de resultats que serveix per indicar els valors resultants de x_train.
- **epoches**: Número de cops que es repetirà l'entrenament dels valors d'entrenament. 1 epoch seria 1 volta als valors, 2 epoches es repetiria 2 cops... fins a n epoches.
- **batch_size**: Número de valors que s'entrenen cada cop. Un valor de 50 batch_size significa que durant l'entrenament s'intenta predir 50 exemples processats a la vegada.
- **validation_data**: Llista de valors que comprovarà el model, després de cada epoch, el valor de pèrdua.
- **callbacks**: Llista de callbacks que s'aplicaran al model durant l'entrenament.
- **verbose**: Variable que indica, en relació al valor, com serà la sortida del com es desenvolupa l'entrenament del model.

4.1.5 Validació model

Finalment, hem de comprovar que el model ja entrenat és capaç de resoldre prediccions fora d'un espai d'entrenament. Per a fer això, usant els fitxers de test, comprovarem el rendiment d'aquest model.

```
def test_loss(model: Sequential, x_test, y_test) -> None:
    loss = model.evaluate(x_test, y_test, verbose=1)
    print(f"\nPérdida en los datos de prueba: {loss}")
```

Amb l'*evaluate*, passant-li els valors de test, ens retornarà el valor de pèrdua. Si es considera que és acceptable, el model es pot guardar. Si no, s'ha de reintentar l'entrenament fent canvis als hiperparàmetres, el tipus de model...

Per a guardar el model, farem un *save* d'aquest. El fitxer resultant serà un .keras. Però, durant l'entrenament, hem fet la normalització de les variables i, en un cas real, s'ha fer la mateixa normalització. Llavors, hem de guardar també els scalers (x_scaler, y_scaler) usant la llibreria *joblib*, específicament la funció dump, que guarda una variable en un fitxer. En el nostre cas, els scalers, cadascun a un fitxer propi .pkl [15].

4.2 Integració de la IA amb el sistema de control

4.2.1 Desenvolupament de la API

Per a poder transformar un codi python en una aplicació API serà necessari usar una llibreria de desenvolupament API. Tenim diferents tipus, com Flask, FastAPI, Django... però usarem en aquest treball FastAPI al ser fàcil de desenvolupar, amb un alt rendiment i orientat exclusivament a desenvolupament de API (Django i Flask està orientat a desenvolupament web). Llavors, per a declarar el programa com a API s'ha d'importar la llibreria i tenir un objecte FastAPI.

```
from fastapi import FastAPI


app = FastAPI()
```

Però una declaració de la aplicació API només crearà el model API. Necessitarem declarar en cada funció que vulguem que es pugui accedir des de fora com s'accedirà a aquesta. Com que només necessitem 1 funció de predicció, només caldrà declarar 1 cop. Per a fer això, la sintaxis és la següent:

```
@app.post("/predict")
```

Com hem dit prèviament, farem la operació HTTP Post i la url ha estat declarada com a */predict*.

4.2.2 Predicció pluja

Per a fer la predicció de la pluja, s'ha creat un fitxer python anomenat "rain_prediction.py". Aquest tindrà tres funcions:

La funció *predict()* serà l'encarregada d'agafar les prediccions resultants i ficar-les en el format correcte. El retorn serà un Dataframe de pandas on els índexs seran els 4 rangs horaris i els valors, la predicció.

```
def predict(self) -> pd.DataFrame:
    """
    Funció que agafa les prediccions futures i les retorna en un format correcte
    :return: Dataframe amb les prediccions futures de pluja en els 4 rangs horaris
    """
    pred = self.get_future()
    return pd.DataFrame(
        data={
            "prediction": [pred.get(p) for p in WeatherPredictor.Period]
        },
```

```
            index=[
                    p.to_datetime().strftime("%Y-%m-%d %H:%M")
                    for p in WeatherPredictor.Period
                ],
        )
```

La funció predict necessita agafar els valors futurs de la pluja i ho fa amb la funció *get_future()*. Aquesta agafarà els valors de les 4 franges horàries i ho transformarà en un diccionari associant les hores amb les prediccions.

```
    def get_future(self):
        """
        Funció que agafarà el que retorna el diccionari de valors amb clau la data
        i el valor de la predicció
        :return: Diccionari amb les prediccions associades a l'hora.
        """
        try:
            pred = {}

            prediction = self._extern_api_req()

            for i, w in enumerate(WeatherPredictor.Period):
                pred[w] = float(prediction[i])

            return pred
        except Exception as e:
            print("Error: " + str(e))
```

Finalment, tenim la funció més important, que es *extern_api_req()*. Aquesta serà la que agafa la predicció de la API externa. Com que és més llarga, la dividiré en 2 parts, declaració per la predicció i resposta.

La URL per a fer la predicció és: `https://api.open-meteo.com/v1/forecast?latitude={latitud}&longitude={longitud}&hourly=precipitation_probability&timezone=Europe%2FMadrid`, on latitud i longitud són els valors de posició.

Llavors, per a fer la crida dins del codi s'ha de fer servir la llibreria requests. El codi resultant és:

```
    response = requests.get(url)      # Enviar la crida a la API
    response.raise_for_status()       # Comprovar que no hi hagi errors
```

Llavors s'ha de comprovar que la resposta sigui vàlida. Per fer-ho, les crides en HTTP sempre tenen un codi associat per indicar com ha anat la sol·licitud. El codi que significa que tot ha anat bé és el 200.

Així, d'aquestes s'han d'agafar les prediccions de pluja que estiguin en les hores que hem concretat. Al ser un json la resposta, agafarem les dates que tinguin valor en les hores 2, 8, 14 i 20.

```
pred = []
for i, time in enumerate(hours):
    if time.startswith(today_date):
        # Canviem el format de data
        hour = datetime.strptime(time, "%Y-%m-%dT%H:%M").hour
        if hour in [2, 8, 14, 20]:
            pred.append(precipitation_prob[i])
return pred
```

Amb això, podrem aconseguir les prediccions del dia i retornar les dades en un format fàcil de manipular per al sistema de control.

4.2.3 Predicció temperatura

Per a predir la temperatura, com s'ha esmenat prèviament, es farà servir el model predictiu de IA. Per a fer això, hem de guardar el model predictiu generat prèviament. Aquest model s'anomenarà "modelo.keras".

Per a fer la predicció, es farà servir una funció que retornarà les prediccions. La funció es dividirà per seccions per aclarir que fa cada part.

```
df = pd.DataFrame([entry.dict() for entry in parameters.data])

input_width = self._predictor.input_width
label_columns = self._predictor.label_columns

current_window = df.iloc[-input_width:].copy()

original_shape = current_window.shape
current_scaled = x_scaler.transform(current_window.values.reshape(-1, original_shape[1]))
current_window = pd.DataFrame(
    current_scaled.reshape(original_shape),
    columns=df.columns
)
```

Agafarem les dades que s'han passat des del sistema de control i agafarem les ultimes 24 dades. Farem una copia d'aquestes dades per evitar qualsevol tipus de problemes de punters.

Com que el model només aconsegueix predir les temperatures dels tancs, tenim 3 dades que són impossibles de poder arribar a predir en un futur, que serien la velocitat del vent, la radiació solar i la radiació infraroja. Per a fer això, usarem dues API externes, Meteo-Sat per les variables de la velocitat del vent i radiació solar i NASA per la radiació infraroja. Llavors, abans de fer cap predicció, tindrem una funció que fa el càlcul dels valors futurs.

```
def get_fut_val():
    a = get_forecast_from_now_local()
    b = get_ir()

    res = []
    for open_data, nasa_data in zip(a, b):
        time = open_data['time']
        wind_speed = open_data['wind_speed']
        solar_radiation = open_data['solar_radiation']
        radiation_infrared = nasa_data['predicted_radiation_infrared']

        res.append({'time': time, 'wind_speed':wind_speed,
        'solar_radiation': solar_radiation, 'radiation_infrared': radiation_infrared})

    return res
```

Aconseguirem les prediccions de futur amb la funció *"get_forecast_from_now_local()"* per les variables de velocitat del vent i radiació solar amb Meteo-Sat i la funció *"get_ir"* per la variable de radiació infraroja amb NASA. La funcionalitat d'aquestes és molt similar a com es crida una API en general. Indicant a les dues API la localització, la franja d'hores (inici i final) de quan es vol buscar les prediccions i les variables que es volen predir a cadascun ens retornarà la predicció corresponent.

Llavors, per cada predicció s'agruparan, fent un zip, les dues respostes de les API en un sol diccionari relacionant el temps amb les variables.

Amb totes les prediccions i tots els temps, podem fer les prediccions. Es farà de la següent forma:

```
predictions = []
        date = datetime.now()

        for i in range(8):
            input_array = np.expand_dims(current_window.values, axis=0)
```

```
tensor = tf.convert_to_tensor(input_array, dtype=tf.float32)

pred = self._predictor.model.predict(tensor, verbose=0).squeeze()
predictions.append(pred)

new_row = current_window.iloc[-1].copy()
print(new_row)
for i, col in enumerate(label_columns):
    new_row[col] = pred[i]

curr = fut_val[i]
year = date.year
seconds_in_day = 24 * 60 * 60
time_seconds = date.hour * 3600 + date.minute * 60 + date.second
new_row["solar_rad_w_m2"] = curr["solar_radiation"]
new_row["ir_rad_w_m2"] = curr["radiation_infrared"]
new_row["wind_vel_m_s"] = curr["wind_speed"]
new_row["day_sin"] = np.sin(2 * np.pi * time_seconds / seconds_in_day)
new_row["day_cos"] = np.cos(2 * np.pi * time_seconds / seconds_in_day)
new_row["year_sin"] = np.sin(2 * np.pi * year / 365)
new_row["year_cos"] = np.cos(2 * np.pi * year / 365)
date = date + timedelta(minutes=15)

current_window = pd.concat([current_window, pd.DataFrame([new_row])],
    ignore_index=True)
current_window = current_window.iloc[1:]
```

Com que la intenció és fer 8 prediccions, es farà un bucle amb les mateixes iteracions. Per a cada una, transformarem la array en un tensor, és a dir, una estructura de dades usada per a aprenentatge automàtic. Llavors, farem la predicció fent la crida al model. Finalment, hem de desplaçar la finestra temporal 15 minuts, ja que si volem predir la temperatura dels tancs en un temps següent, necessitarem les prediccions 6 hores passades a l'hora corresponent. Per a fer això, la temperatura que hem predit serà la ultima finestra en el moment posterior, eliminant el primer valor de la finestra. De forma gràfica, té aquesta representació.

T-24	T-23	T-22	...	T0
...	...	...	...	...

Table 1: Finestra predicció t0 + 15 min

Canvi finestra $\longrightarrow$

T-23	T-22	...	T0	T1
...	...	...	...	...

Table 2: Finestra predicció t0 + 30 min

Figure 6: Desplaçament de la finestra temporal amb predicció autoregressiva

Finalment, es desnormalitzaran els valors de les dades en els valors reals.

4.2.4 Predicció demanda

Com s'ha dit al disseny, la predicció de la demanda es farà mitjançant la lectura d'un històric de dades de tot l'any. En el nostre cas, aquest fitxer serà el "demand-house". Com s'ha esmentat a la part del disseny, aquest fitxer contindrà les 3 columnes del nostre interès, Date/Time, Heating i Cooling i cada fila contindrà els valors de la demanda en el corresponent moment temporal.

Llavors, el tractament del fitxer s'ha realitzat de la següent forma:

```
def _read_demand_csv(path: str, date_format: str) -> pd.DataFrame:
    df = pd.read_csv(
        path,
        sep=r",",
        header=0,
        engine="python",
        encoding="latin1",
    )
    df["datetime"] = pd.to_datetime(
        df["Date/Time"].apply(_to_datetime),
        format=date_format,
    )
    return (
        df.drop(columns=["Date/Time"])
        .set_index("datetime")
        .rename(
            columns={
                "Heating [J]": "heating",
                "Cooling [J]": "cooling",
            }
        )
    )
```

Primer de tot, s'ha de llegir el fitxer amb les dades. Per a fer això, farem servir la funció de pandas *read_csv* que ens retorna un Dataframe amb totes les dates. Llavors, arreglarem el format del camp Date/Time per a que sigui del format correcte i retornarem la llista arreglant els noms de les columnes a uns de més fàcils. Amb això tenim la lectura completa del fitxer, ara hem de seleccionar el rang de demanda que volem. Per això, farem servir la següent funció:

```
def predict(self, parameters: pd.DataFrame) -> pd.DataFrame:
    """
    Funció que predeix la demanda futura
    :param: Rang de temps (inici i final) de la demanda que es vol agafar
    :return: Predicció de la demanda
    """
    start = datetime.datetime.strptime(
        parameters["start"].to_numpy()[0],
        GLOBAL_DATE_FORMAT,
    )
    end = datetime.datetime.strptime(
        parameters["end"].to_numpy()[0],
        GLOBAL_DATE_FORMAT,
    )
    df = self._demand[["cooling", "heating"]]
    return df.loc[(start <= df.index) & (df.index <= end)]
```

Per a fer-ho, transformarem l'inici i final que s'ha passat per paràmetres en format de data igual al que s'ha transformat prèviament el fitxer i agafarem amb la funció "loc" aquells que estiguin dins del rang.

Amb això retornarem totes les dates dins del rang. Com que el format de sortida és en Joules, no cal fer cap transformació addicional ja que es podria comparar amb la producció de forma directa, perquè també és una predicció en Joules.

4.2.5 Predicció producció

Per a predir la producció, farem servir la formula que s'ha explicat durant el disseny. Com que tenim els dos tipus de temperatures, s'haurà de calcular la producció en els dos tipus i retornar un diccionari amb totes les prediccions.

```
def predict(self, data: DataFrame) -> DataFrame:

    data["prod_cold"] = data.iloc[1:].apply(
        lambda x: Ei(
```

```
            self._specs,
            t_0=data.iloc[0]["cold"],
            t_1=x["cold"],
            mode="cold",
        ),
        axis=1,
    )

    data["prod_hot"] = data.iloc[1:].apply(
        lambda x: Ei(
            self._specs,
            t_0=data.iloc[0]["hot"],
            t_1=x["hot"],
            mode="hot",
        ),
        axis=1,
    )

    return (
        data.iloc[1:]
        .drop(columns=["cold", "hot"])
        .rename(
            columns={
                "prod_hot": "hot",
                "prod_cold": "cold",
            }
        )
    )
```

Llavors, per cada temps de cada temperatura calcularem la potencia generada usant la formula. Passarem el mode que serà hot o cold, i els paràmetres com el cabal és passarà mitjançant una classe auxiliat anomenat RceSpecs.

```
def Ei(
    rce: RceSpecs,
    t_0: np.float16,
    t_1: np.float16,
    mode: Literal["hot", "cold"] = "hot",
) -> np.float16:
```

```
    v = rce.VH if mode == "hot" else rce.VC
    return v * rce.RHO * rce.CP * (t_1 - t_0)
```

4.2.6 Retorn de la API

Un cop ja tenim les 4 prediccions, podem retornar al Sistema de Control tot el que s'ha anat predient. Per a fer això, s'ha desenvolupat una classe que facilitarà agrupar les dades en un sol fitxer. Aquesta serà la funció OutputBuilder, que segueix el patró de disseny Builder, que contindrà un diccionari amb totes les futures dades. S'ha fet així per a poder arribar a escalar la API si en un futur s'hi vol afegir més prediccions. Aquesta classe tindrà una funció d'afegir data, que incorporarà tot el que s'ha predit dins d'una clau del diccionari.

Llavors, amb aquesta classe agruparem totes les prediccions.

```
out.add_data(
            "rain-prediction",
            structure.rain(r_pred),
    ).add_data(
            "demand",
            structure.dema(d_pred, _future),
    ).add_data(
            "energy-production",
            structure.prod(
                p_pred,
                _future,
                run_config["temperature"]["labels"],
            ),
    ).add_data(
            "tank-temperature",
            structure.temp(
                t_pred,
                _future,
                run_config["temperature"]["column-indices"],
                run_config["temperature"]["labels"],
    )
```

Llavors, amb aquest codi podem veure que el diccionari contindrà 4 claus, una per cada predicció, amb el nom "rain-prediction", "demand", "energy-production" i "tank-temperature". Com que aquesta classe segueix el patró Builder, serà necessari retornar el diccionari amb la funció *build*, que generarà una copia del diccionari.

4.3 Desenvolupament del sistema de control

4.3.1 Comentaris previs

El Sistema de Control ha estat dissenyat i desenvolupat amb codi font per a poder comprovar la robustesa d'aquest. Durant el desenvolupament, s'ha tingut molt en compte el disseny d'aquest perquè pugui assolir uns bons resultats i s'ha buscat que d'integració i comunicació entre els diferents components es faci de forma eficient i sense errors. No obstant això, per la part de validació s'ha quedat fora de l'abast del projecte la integració en el prototipus real. Això significarà que per poder veure si el model funciona, es farà ús d'arxius amb dades en lloc de lectures reals.

4.3.2 Lectura dades

Aquest component no s'ha pogut desenvolupar a causa de la falta de temps i una connexió als sensors o una base de dades amb dades actuals. Llavors, s'explicarà la intenció i que s'ha desenvolupat per poder arribar a provar el sistema.

- Les dades s'havien de connectar en una base de dades o fitxer amb dades que va guardant el que els sensors de la màquina van detectant. Llavors, de forma automàtica el sistema havia de llegir els últims valors necessaris. Finalment, aquestes dades es transformaran al format que l'API necessita.
- Per a poder agafar les dades de forma que el sistema funcioni correctament, tenim un fitxer de test que tindrà les dades prèvies a l'hora actual però en un any anterior.

Per agafar les últimes dades a l'hora prèvia d'ara, tindrem una funció externa de prova que agafarà les 24 últimes dades.

```
df = pd.read_csv("cleansed.csv")
df["datetime"] = pd.to_datetime(df["datetime"])

df = df[df["datetime"] < fecha_objetivo].tail(24)

df = df.to_dict(orient="records")

df = process_data(df)
out = {"data": df}
json_data2 = json.dumps(out, indent=4)

with open('test.json', 'w', encoding='utf-8') as json_file:
    json_file.write(json_data2)
```

Llavors, es crearà el fitxer *test.json* que contindrà les dades per la predicció de temperatura. Llavors, el sistema de control farà la funció get_data() per llegir-ho.

```
def get_data():
    with open(r"Control_system\src\api\test.json") as p:
        j = json.load(p)
    return j
```

4.3.3 Elecció Finestra Temperatura Calenta

En el cas de la temperatura calenta, es farà el càlcul a les 7:15 del matí, que aquesta hora és la mitjana durant tot l'any en què surt el sol. Llavors, seleccionarem la demanda predita que sigui des del migdia del dia 1 fins al matí del dia 2.

```
    start_dem_heat = now.replace(hour=12, minute=0, second=0, microsecond=0)
    end_dem_heat = (now + timedelta(days=1))
        .replace(hour=12, minute=0, second=0, microsecond=0)
    dem_c = calculate_dem_for_period(dem_data['heating'], start_dem_heat, end_dem_heat)
```

Per tant, un cop tenim la demanda necessària, hem de trobar en quines finestres de temps la producció és major. Per això, farem servir la funció *calculate_optimal_production_plan()*

```
def calculate_optimal_production_plan(available_prod, target_demand, type):
    sorted_prod = dict(sorted(available_prod.items(), key=lambda item: item[1]))

    accumulated_production = 0.0
    selected_frames = []
    for time_frame_str, production_value in sorted_prod.items():
        accumulated_production += float(production_value)
        selected_frames.append(time_frame_str)
        if accumulated_production >= target_demand:
            break

    if accumulated_production < target_demand:
        return [], -1

    selected_frames_dt = [datetime.fromisoformat(ts) for ts in selected_frames]

    for time in selected_frames_dt:
        temp_mode[time] = type
    return selected_frames_dt, accumulated_production
```

Ordenarem la llista de prediccions de més a menys producció i agafarem finestres de temps fins que arribem a la demanda. Si no s'arribés, es retornaria error indicat amb el valor -1. A més a més, per cada hora seleccionada es guardarà el mode del tanc.

4.3.4 Elecció Finestra Temperatura Freda

El funcionament és el mateix que el de la temperatura calenta, però en lloc de calcular-ho a la sortida del sol es farà a la posta. L'hora mitjana és a les 19:15. Llavors, el procediment és el mateix, calcular la demanda prevista però de tot l'endemà i trobar la finestra de producció.

```
start_dem_cool = (now + timedelta(days=1))
    .replace(hour=0, minute=0, second=0, microsecond=0)
end_dem_cool = (now + timedelta(days=2))
    .replace(hour=0, minute=0, second=0, microsecond=0)
dem_f = calculate_dem_for_period(dem_data['cooling'],
    start_dem_cool, end_dem_cool)

selected_time_frames, total_predicted_production =
    calculate_optimal_production_plan(prod_data['cold'], dem_f,0)
```

4.3.5 Decisions

Aleshores, en cada *time_step*, comprovarem si en aquell moment s'està en la finestra de control. Tindrem dos casos:

- **Estem dins de la finestra**: Es calcularà l'índex de rendiment (COP). Si aquest índex és major a 1, s'activarà la màquina. Si és menor, es tancarà assumint que no podem suplir la demanda.
- **No estem dins de la finestra**: Es mantindrà la bomba tancada.

Tot això es farà amb la següent lògica de codi:

```
current_frame_str = time_frame_dt.isoformat(timespec='seconds')
mode = temp_mode[now]
if mode == "hot" and current_frame_str in prod_data['hot']:
    current_frame_prod_value = float(prod_data['hot'][current_frame_str])
    print(f"{now}: Producció de calor prevista per a aquesta franja:
        {current_frame_prod_value:.2f} Wh.")
elif mode == "cold" and current_frame_str in prod_data['cold']:
    current_frame_prod_value = float(prod_data['cold'][current_frame_str])
    print(f"{now}: Producció de fred prevista per a aquesta franja:
```

```
            {current_frame_prod_value:.2f} Wh.")
    else:
        print(f"{now}: No s'han trobat dades de producció per a la franja actual
            en el mode {mode}.")

    energy_consumed_pump_wh = P_bomba_watts * time_step_hours
    print(f"{now}: Energia que consumirà la bomba si està ON:
        {energy_consumed_pump_wh:.2f} Wh.")
    if energy_consumed_pump_wh > 0:
        COP = current_frame_prod_value / energy_consumed_pump_wh
        print(f"{now}: COP calculat: {COP:.2f}.")
    else:
        COP = 0
        print(f"{now}: Energia consumida per la bomba és zero, COP establert a 0.")

    if COP >= 1:
        action = 1
        print(f"{now}: COP és >= 1. Bomba ON.")
    else:
        action = 0
        print(f"{now}: COP és < 1. Bomba OFF.")
    break
```

Hi ha una variable global *action* que indicarà si la màquina s'ha d'obrir o no. Aquesta la declarem a 1 si COP és favorable i 0 si no.

4.3.6 Ajustar prediccions

Al final d'un *time_step*, s'ha de comprovar que el que s'ha predit s'ha complert. Per a fer això, arreglarem la demanda restant-li la potència generada. Si aquesta és la mateixa, es mantindrà la finestra de control.

```
    if current_dem_target > 0:
        old_dem_target = current_dem_target
        current_dem_target = verify_and_adjust_demand(current_dem_target)
        print(f"{now}: Demanda restant ajustada de {old_dem_target:.2f} Wh
            a {current_dem_target:.2f} Wh.")
        if current_dem_target <= 0:
            selected_time_frames = []
            total_predicted_production = 0.0
```

```
            current_dem_target = 0.0
            action = 0
            print(f"{now}: Demanda coberta o esgotada. Reiniciant pla i bomba OFF.")

    else:
        action = 0
        print(f"{now}: No hi ha demanda objectiu pendent. Bomba OFF.")
```

Aquesta funció, en un futur, s'ha d'acabar millorant, ja que, com que no es pot comprovar la potència real generada, no es pot assegurar que la predicció tingui un correcte funcionament.

5 Experimentació i resultats

Finalment, un cop desenvolupat el model, s'ha de comprovar que tot el que s'ha realitzat té uns resultats vàlids. Per a fer això, s'ha ideat un conjunt de proves per a assegurar el correcte funcionament de no només el sistema de control sinó de també el model predictiu i la API.

5.1 Metodologia

5.1.1 Model IA

Per a provar el model predictiu, farem dos tipus de proves diferents. Un tipus de prova que podem fer es variar la mida de la finestra. Es farà en tres espais de temps diferents, una amb la mida de finestra de 2 hores representant prediccions a curt termini, una amb la mida de 6 hores representant prediccions a un termini mitjà i una amb 24 hores i 48 hores per veure com evoluciona el model en un temps alt.

Un altre tipus de prova és realitzar el mateix tipus de mida de finestra en diferents èpoques de l'any, ja que el comportament de la temperatura no és igual a l'hivern que a l'estiu, per exemple. Per això, s'ha de provar en tot l'any. Per a fer-ho, es farà una prova en cada estació de l'any en la finestra de 48 hores, on es pot comprovar el rendiment en un temps llarg.

Però, comprovar en un dia aleatori i que el model funcioni no assegura que en general aquest realment funcioni correctament. Per a fer-ho més vàlid, cada prova s'ha realitzat 50 cops seguits en el mateix instant de temps. Amb això ens assegurem que, de mitjana, els valors siguin vàlids.

5.1.2 API

Per fer les proves d'aquest model, farem servir fitxers JSON de prova per a comprovar quines prediccions ens retornen. Llavors, utilitzarem l'aplicació Postman per fer les diferents crides.

Per a poder realitzar qualsevol model, es crearà un programa extern que agafarà les ultimes 24 dades prèvies a l'hora que es volia. En aquest, es llegirà tot el fitxer de proves i s'escolliran les últimes 24 proves.

5.1.3 Sistema de Control

Finalment, per comprovar el funcionament del sistema de control, observarem el que va executant el sistema, mitjançant missatges per la sortida estàndard a terminal.

Com passava amb l'API, les dades no són reals, la qual cosa comporta que no podrem arribar a validar-ho en un entorn real. Per això, s'interpretaran i explicaran els resultats obtinguts. Per aconseguir les dades, es tornarà a usar el programa extern prèviament nomenat a l'apartat 6.1.2 de l'API.

5.2 Resultats

5.2.1 Model IA

A l'hora de fer proves, es pot fer de forma numèrica indicant l'error entre temperatures real i predita. Però a l'hora de comprovar el rendiment, és força més difícil de veure. Per això, es faran servir gràfiques per a representar el comportament de les temperatures. Com s'ha dit a l'apartat de proves, farem proves en diferents rangs de temps.

Es representarà l'evolució de les temperatures mitjançant 4 línies diferents. La línia blava representarà la temperatura calenta real, la groga la temperatura calenta predita, la verda la temperatura freda real i la vermella la temperatura freda predita. Cada punt representarà la temperatura en un temps. Cada temperatura tindrà la marca de temps a l'eix de les x en el format dia i mes i l'hora i minut de la predicció.

Finestra de 2 hores

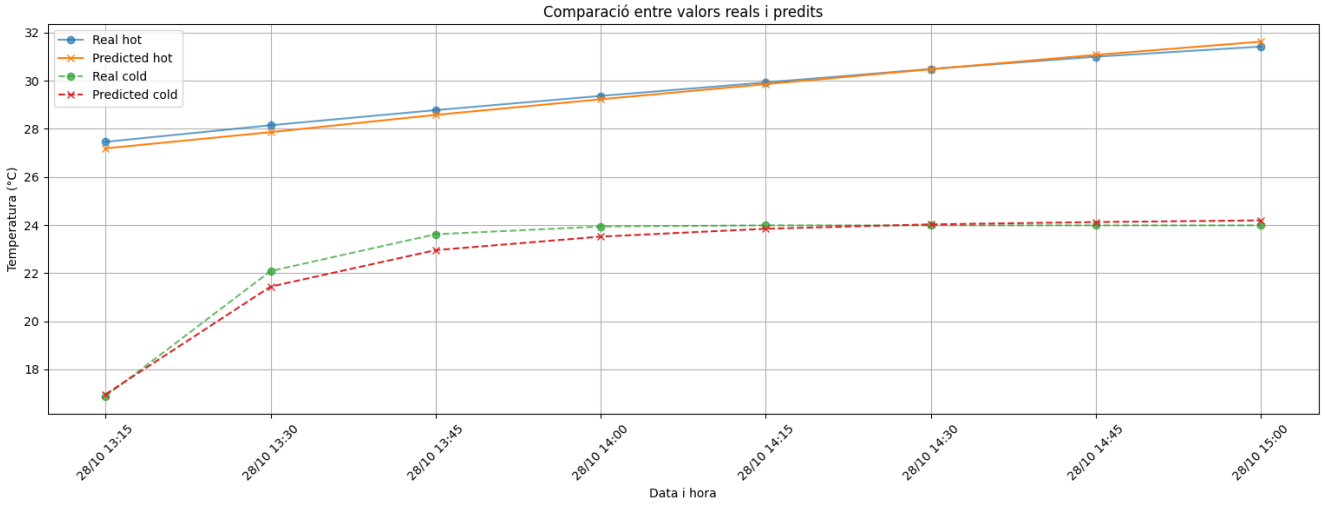

Figure 7: Comportament temperatura en 2 hores

Es pot comprovar que en un rang de 2 hores, la diferència entre les rectes és força mínima, ja que segueixen un comportament similar. Malgrat això, es pot veure que en canvis més grans, entre les 13:15 i 14:00, la diferència augmenta, encara que aquesta no supera ni el grau.

Finestra de 6 hores

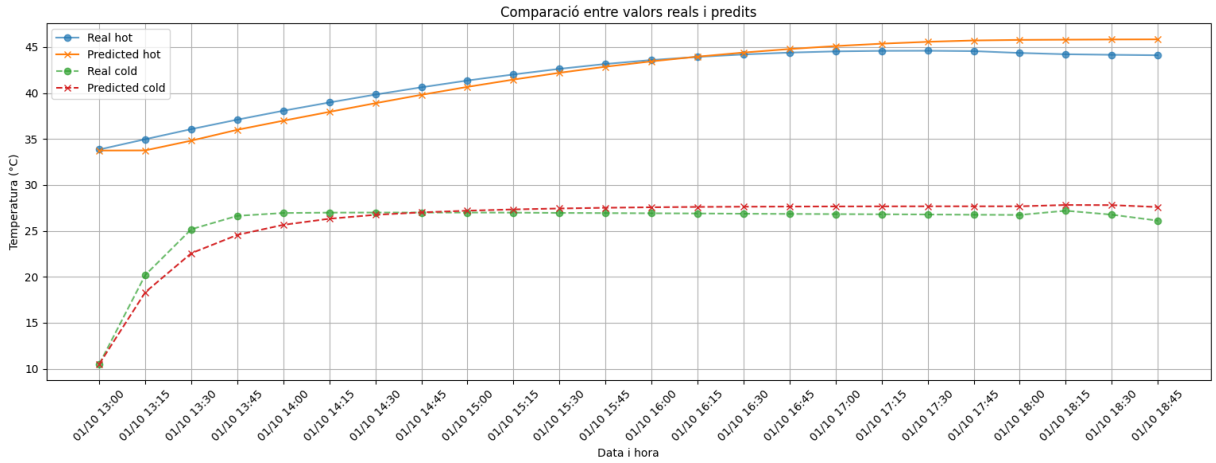

Figure 8: Comportament temperatura en 6 hores

Augmentant la finestra podem veure que les prediccions en general continuen tenint un comportament molt similar a les reals. Malgrat això, també podem veure un augment de la diferència quan el comportament de les temperatures creix o decreix de forma abrupta. Però, encara que no arribi a tenir un comportament tant radical la predicció com el real, podem veure que el comportament d'aquest continua sent similar al real.

Finestra de 24 hores

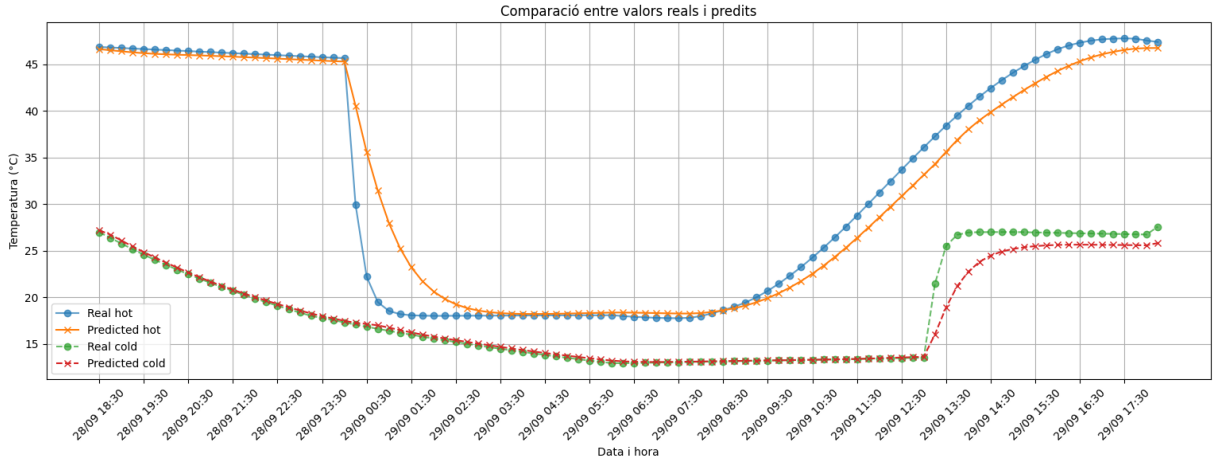

Figure 9: Comportament temperatura en 24 hores

Podem comprovar quin es el funcionament en un rang de finestra molt gran. En el cas de les 24 hores, és fàcil veure que la tendència de les prediccions continua sent similar a la dels valors reals. Podem veure que quan hi ha un canvi de temperatura gran, la diferència s'accentua molt, ja que

el model no arriba a ser tant ràpid al canvi i té una reducció o un augment de temperatura més gradual. **Finestra de 48 hores**

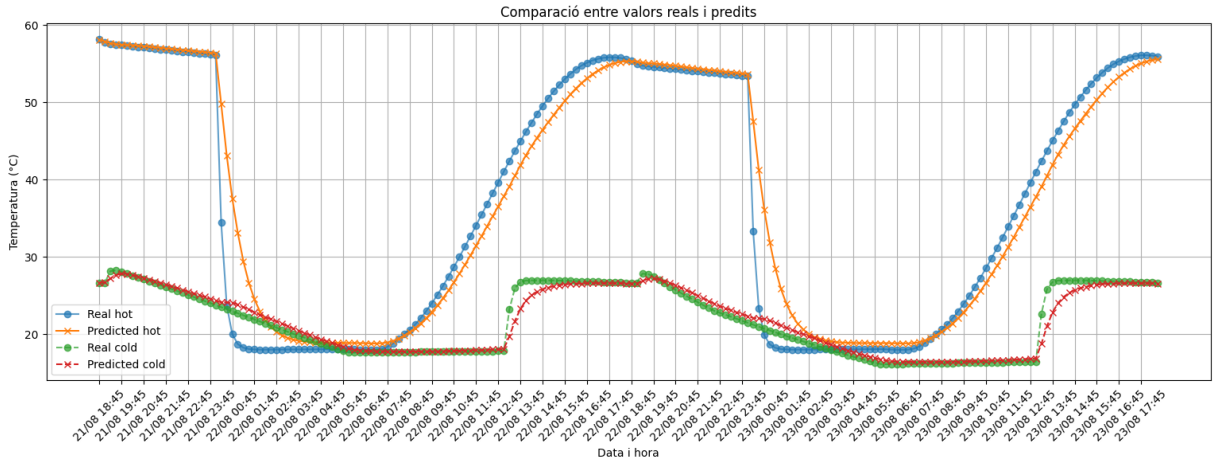

Figure 10: Comportament temperatura en 48 hores

En un rang de 48 hores podem comprovar que les tendències es continuen mantenint encara que se segueixen mostrant valors molt diferents en moments puntuals.

Prediccions per diferents estacions

Com s'ha comentat prèviament, s'ha de comprovar que el model s'adapta en diferents moments de l'any, ja que durant l'estiu les temperatures dels tancs no tindran el mateix comportament. Per a fer això, comprovarem el rendiment durant quatre dies, cada una a cada estació, per veure si continua realitzant bones prediccions.

Predicció Estiu

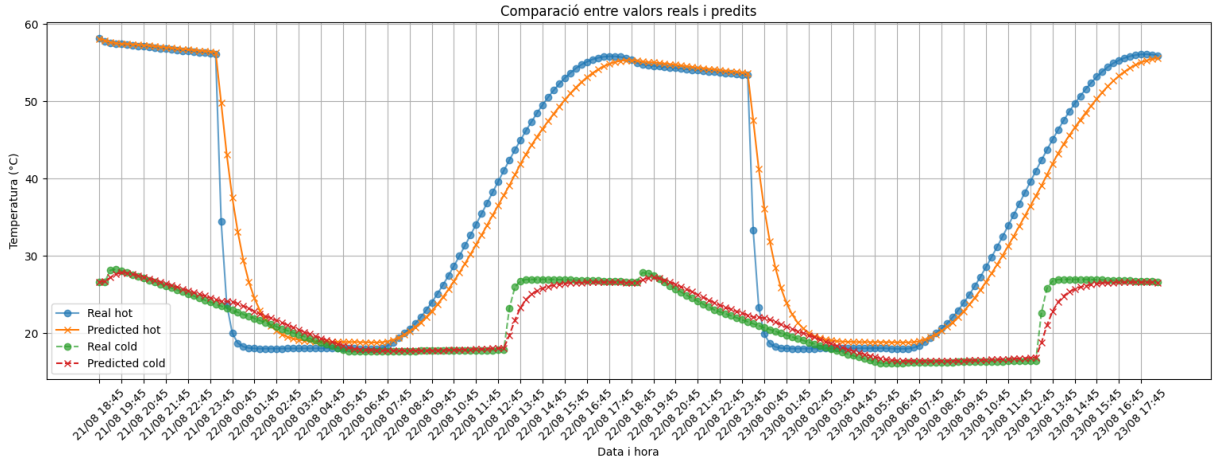

Figure 11: Predicció Estiu

Aquesta és la mateixa representació que durant les 48h. Es pot veure que amb temperatures altes d'estiu, com que la difèrencia és força gran, pateix més quan hi ha canvis més sobtats. En les temperatures fredes, com que fins i tot a la nit no hi ha un ambient gaire fred, la temperatura no fluctua de forma tan ràpida.

Predicció Tardor

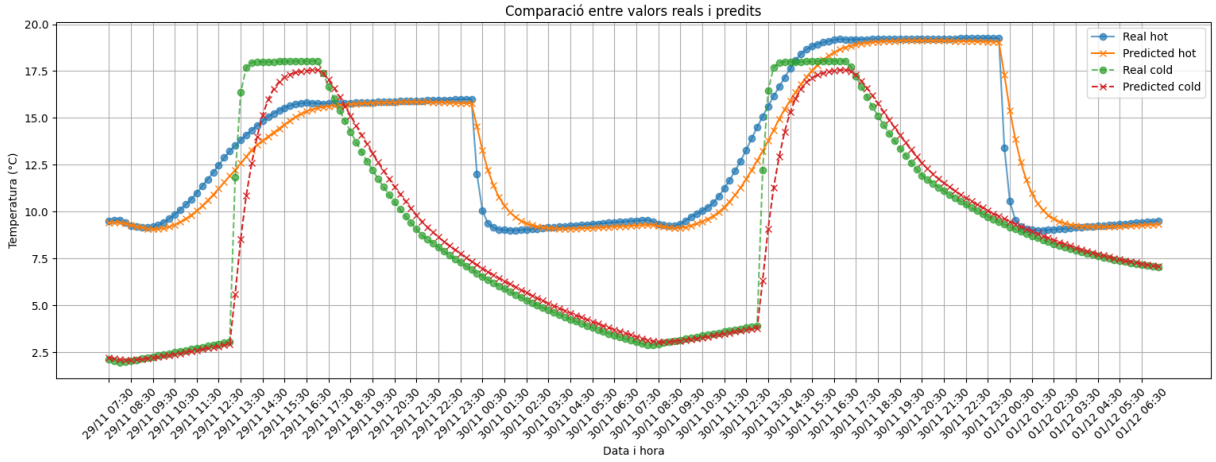

Figure 12: Predicció Tardor

Aquí es pot veure que durant la tardor les temperatures fredes i calentes estan en un entorn més similar. Com que cada cop fa més fred, el canvi de temperatura es nota més i hi ha més error

a la predicció. A la predicció de calor, és a la inversa, ja que a l'haver un clima menys calent la temperatura baixa.

Predicció Hivern

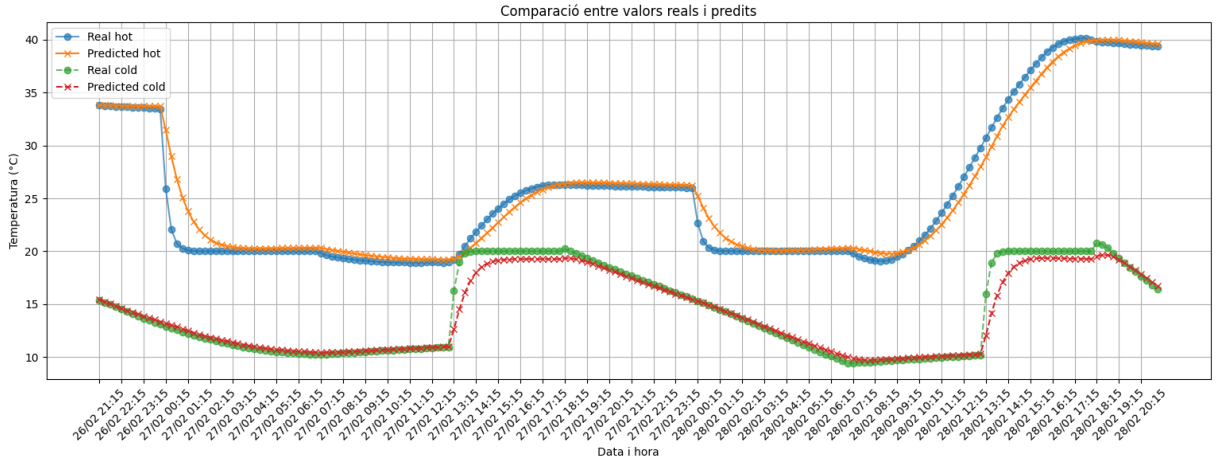

Figure 13: Predicció Hivern

En l'hivern, es pot veure que la diferència de temperatures fredes entre la màxima i la mínima és ja força gran, mentre que de la calor és molt reduïda en comparació amb l'estiu.

Predicció Primavera

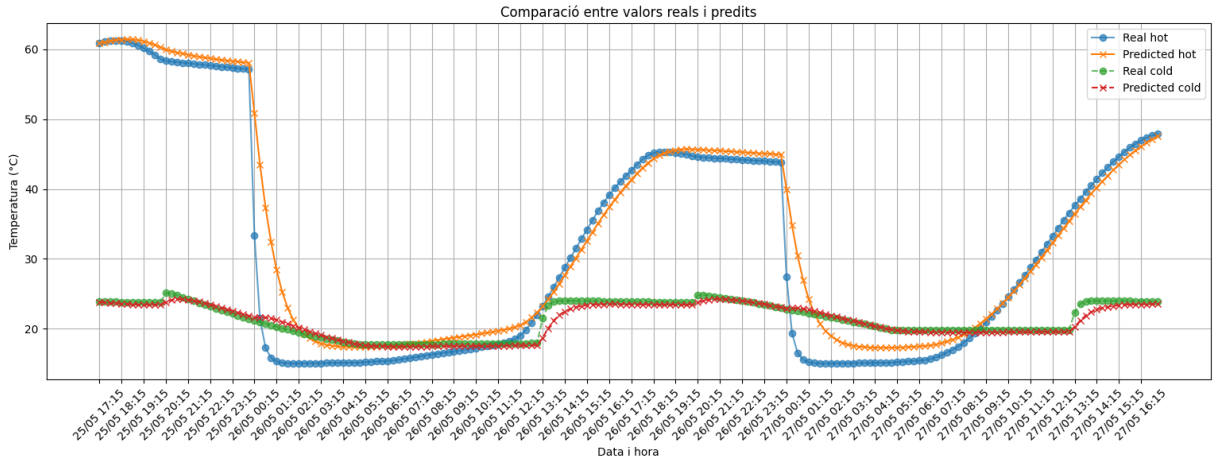

Figure 14: Predicció Primavera

Finalment, en la primavera, es pot veure que la calor torna a pujar força ja que s'apropa a l'estiu i la diferència torna a pujar, mentre que la temperatura freda comença a reduir-se.

5.2.2 API

Com s'ha comentat, per a fer les proves s'ha fet servir el programa Postman [20]. Aquest permetrà fer crides a l'API directament, sense haver d'usar codi font. Llavors, per tal de fer una prova, crearem una nova *request* i, en l'URL, definirem la crida com a POST i escriurem el nostre URL.

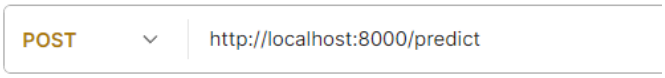

Figure 15: Request URL

Llavors, hem de passar les 24 ultimes dades. És farà com s'ha comentat prèviament durant el disseny de la API, a la pàgina 17. En Postman, s'inserirà el JSON en la part de la request body.


```
{
    "data": [
        {
            "cold": 14.49,
            "hot": 20.3,
            "wind_vel_m_s": 0.863,
            "solar_rad_w_m2": 0.0,
            "ir_rad_w_m2": 327.848,
            "day_sin": 0.9469301294951056,
            "day_cos": 0.3214394653031617,
            "year_sin": 0.8263541987239096,
            "year_cos": -0.5631507242749186
        },
        {
            "cold": 14.29,
            "hot": 20.3,
            "wind_vel_m_s": 0.887,
            "solar_rad_w_m2": 0.0,
            "ir_rad_w_m2": 327.559,
            "day_sin": 0.9659258262890683,
            "day_cos": 0.25881904510252074,
            "year_sin": 0.8263541987239096,
            "year_cos": -0.5631507242749186
        },
```


Figure 16: Request body

Llavors, hem de comprovar que funciona correctament. En el cas que hi hagi un error, es retornarà un JSON d'error. Per a poder provar el que retorna, ho comprovarem enviant un JSON amb un valor diferent de 24 temps. El *body* tindrà la següent forma:

```
{
    "data": [
        {
            "cold": 14.49,
            "hot": 20.3,
            "wind_vel_m_s": 0.863,
            "solar_rad_w_m2": 0.0,
            "ir_rad_w_m2": 327.848,
            "day_sin": 0.9469301294951056,
            "day_cos": 0.3214394653031617,
            "year_sin": 0.8263541987239096,
            "year_cos": -0.5631507242749186
        }
    ]
}
```


Llavors, el JSON resultat hauria de ser un d'error. En aquest cas, et retorna el següent document:


```
{
    "info": {
        "explanation": "Fetched data is not the expected length (expected: 24 != 1)"
    }
}
```


En el cas que l'API funcioni correctament, comprovarem els resultats. Com que no podem comparar els valors amb uns de reals, validarem si els valors resultants tenen un resultat lògic. Com que la resposta és molt gran, la dividirem per cada tipus de predicció. Les 4 prediccions estaran dins de la variable "info".


```
{
    "info": {
        "rain-prediction":{...},
        "demand":{...},
        "energy-production":{...},
        "tank-temperature":{...}
    }
```


Llavors, l'index de pluja retornarà dades de la següent forma:


```
    "rain-prediction": {
        "prediction": {
```

```
            "2025-07-02 02:00": 0.0,
            "2025-07-02 08:00": 0.0,
            "2025-07-02 14:00": 0.0,
            "2025-07-02 20:00": 0.0
        }
```


Com es mostra, en tot el dia s'indica l'absència de pluja. Això és coherent amb les condicions meteorològiques observades en els dies observats, ja que no s'ha registrat episodis de pluja el dia 2 de juliol.

Observant la demanda tenim els següents resultats:


```
        "demand": {
            "cooling": {
                "2025-07-02T17:19:00": 248193.94371389,
                "2025-07-02T17:20:00": 248433.869804101,
                "2025-07-02T17:21:00": 248668.917944556,
                "2025-07-02T17:22:00": 248898.59457879,
                ...
            },
            "heating": {
                "2025-07-02T17:19:00": 0.0,
                "2025-07-02T17:20:00": 0.0,
                "2025-07-02T17:21:00": 0.0,
                "2025-07-02T17:22:00": 0.0,
                ...
            }
        },
```


Aquest resultat, com que és una lectura d'un fitxer, no podem comprovar directament si està o no correcte. Malgrat això, usant la lògica, podem suposar que aquesta demanda pot arribar a ser possible, ja que en l'estiu la demanda de fred és constant i la demanda de calor no sol ser necessària.

Observant la producció, comprovarem els valors:


```
    "energy-production": {
        "cold": {
            "2025-07-02 17:19": "-1239280.27",
            "2025-07-02 17:34": "-1665357.49",
            "2025-07-02 17:49": "-2018678.3",
            "2025-07-02 18:04": "-2044962.25",
```

```
            ...
        },
        "hot": {
            "2025-07-02 17:19": "19847922.61",
            "2025-07-02 17:34": "19643273.09",
            "2025-07-02 17:49": "20011429.94",
            "2025-07-02 18:04": "19723944.35",
            ...
        }
    }
```


Com podem veure, és normal que la producció en calor sigui positiva durant la tarda mentre que la de fred sigui més baixa.

Finalment, hem de comprovar que la temperatura també tingui valors reals.


```
"tank-temperature": {
    "hot": {
        "2025-07-02 17:19": "30.447577",
        "2025-07-02 17:34": "30.80758",
        "2025-07-02 17:49": "31.025047",
        "2025-07-02 18:04": "31.450571",
        "2025-07-02 18:19": "33.754192",
        ...
    },
    "cold": {
        "2025-07-02 17:19": "14.087403",
        "2025-07-02 17:34": "12.054592",
        "2025-07-02 17:49": "10.368901",
        "2025-07-02 18:04": "10.243501",
        "2025-07-02 18:19": "10.236158",
        ...
    }
}
```


Aquestes temperatures segueixen una tendència que pot resultar lògica en relació amb el temps. Com que el dia de l'elecció és un dia de juliol, la temperatura és alta durant bona part del dia. Per això, que la temperatura calenta pujo i la freda baixi a les 5 és realista.

Aquí només s'ha mostrat una única prova, però de mitjana la tendència dels tres era molt similar al del dia mostrat.

5.2.3 Sistema de Control

Per a provar el sistema de control, s'haurà d'executar la següent línia des d'una consola o un IDE amb una consola integrada:

```
python ./Control_system/src/main.py
```

Com que actualment no està integrada la creació d'imatges en Docker, hi ha una dependència cap a l'API. És per això que, abans d'executar la línia prèvia, s'haurà d'iniciar l'aplicació API amb la següent línia:

```
python ./RCE_predictor/api/main.py
```

Llavors, un cop en execució els dos projectes es mostrarà unes prediccions inicials.

```
2025-07-03 17:28:03: Sistema iniciat. Carregant dades inicials.
2025-07-03 17:28:22: Prediccions inicials carregades. Dia de darrera actualització: 3
2025-07-03 17:28:22: Planificació inicial de la demanda de calor.
2025-07-03 17:28:22: No hi ha demanda de calor en els propers dies.
2025-07-03 17:28:22: Planificació inicial de la demanda de fred.
2025-07-03 17:28:22: Demanda de fred estimada: 74730529.41 Wh. Producció prevista per cobrir-la:
2025-07-03 17:28:22: No s'ha pogut trobar un pla òptim per a la demanda de fred. Demanda: 7473052
```

Com es pot comprovar, el primer que es fa en el sistema és generar unes finestres de control principals. Es farà una crida a l'API i s'iniciarà la planificació de la demanda. En el cas de la calor, com que no hi ha demanda de calor, el sistema no necessitarà obrir en temperatura alta i, per tant, no s'afegirà cap temps a la finestra de control. En el cas del fred, la demanda de fred estimada supera la producció que es pot generar i, en conseqüència, no s'obrirà el sistema indicant-ho amb un -1.

Llavors, per cada *time_step*, es mostrarà l'execució en el temps que es realitza.

```
2025-07-03 17:28:22: Nou cicle de control.
2025-07-03 17:28:22: Lògica de control en temps real. Demanda objectiu actual: 74730529.41 Wh.
2025-07-03 17:28:22: Franja horària actual a avaluar: 2025-07-03T17:15:00 a 2025-07-03T17:30:00
2025-07-03 17:28:22: L'hora actual no està dins de les franges seleccionades. Bomba OFF.
2025-07-03 17:28:22: Estat final de la bomba per a aquest cicle: OFF.
```

En cada cicle de control, es comprovarà si s'està dintre d'una franja horària. En aquest cas, la franja és de les 17:15 a les 17:30. Llavors, com que no està dintre d'una franja, la màquina es manté tancada.

5.3 Discussió resultats

Un cop dutes a terme totes les proves, hem pogut comprovar que, en general, els resultats són bons. El model d'IA ha mostrat adaptar-se en diferents entorns i situacions simulades, retornant una resposta coherent davant variacions en les condicions d'entrada.

Pel que fa a l'API, aquesta aconsegueix retornar totes les 4 prediccions en un temps raonable i dins dels límits requerits pel seu ús. Això permet que es pugui integrar al sistema de forma fàcil sense provocar colls d'ampolla.

El sistema de control pot trobar les millors finestres i comprovar si s'obre o no la màquina. La lògica de decisió basada en prediccions ha sigut molt efectiva, les decisions són potencialment eficients.

Llavors, en un entorn de proves s'ha comprovat un correcte funcionament, que era un dels objectius principals del projecte. Per adaptar aquest model al sistema real, serà necessari fer unes millores al sistema, com canviar com es llegeixen les dades i afegir la comprovació de l'energia consumida. Tot això és desenvoluparà en un treball futur.

6 Conclusions

6.1 Conclusions finals

En aquest treball s'ha demostrat que integrar un model d'IA en sistemes de control com el RCE representa una millora en el rendiment i eficiència d'aquest. El sistema dissenyat és capaç de prendre decisions gràcies a prediccions futures, de forma modular, escalable i adaptable.

S'ha validat que el model pot arribar a predir amb precisió temperatures futures, adaptant-se a escenaris canviants. Aquest model junt amb una API facilita la comunicació entre els diferents components del sistema, cosa que millora la mantenibilitat i portabilitat del sistema.

En resum, en aquest projecte s'han pogut assolir els diferents objectius marcats. A més a més, s'ha demostrat el potencial de combinar la tecnologia amb els models d'IA, que està cada cop més integrada a la societat actual. Lluny de ser un problema, la IA és una eina molt poderosa i eficient, capaç d'oferir solucions a reptes tecnològics complexos.

6.2 Limitacions

Com s'ha anat comentant durant el treball, l'estimació de temps en les diferents seccions del projecte ha estat molt diferent de com estava planejat. En un primer moment, es va estimar que el desenvolupament de l'API seria d'un mes aproximadament, però com que vaig haver de refer tot el codi, amb el model d'IA inclòs, es va allargar molt més del que pensava. Això ha comportat diferents limitacions sobretot al sistema de control.

- **Sistema de control amb components no desenvolupats**: Com el sistema de control s'ha realitzat amb molt poc marge de temps, hi ha hagut components que no s'han pogut desenvolupar com estava previst. Per exemple, la forma d'agafar les dades no és la que hauria de ser, no és té en compte la producció real en tenir dades molt del passat...
- **Joc de proves molt limitat**: A causa de l'absència de dades recents, no es podia provar si els components del sistema de control, en casos més actuals, actuaran també de forma correcta.
- **Codi inicial de la API**: El programa de l'API que estava desenvolupat en un principi no funcionava correctament, ja que en intentar executar el codi donava problemes. Es va invertir força temps a recrear el model fent els canvis necessaris perquè finalment retornés dades útils.
- **Fitxers de log**: Actualment, la forma de comprovar el funcionament real és únicament amb execució de codi, cosa que en cas de voler comprovar un error, s'ha de repetir aquest durant l'execució.

6.3 Línies de recerca futures

El sistema de control actualment funciona en un entorn de proves, però no vol dir que aquest acabi funcionant en un entorn real. Cara a futur, un dels passos més important serà traslladar aquest sistema que funciona cap al prototip real, on es connectarà amb el PLC o una base de dades que guarda les dades dels sensors per poder validar en un entorn real. A més a més, mitjançant la coordinació del grup CREA s'anirà validant si el sistema funciona correctament.

S'ha de realitzar un seguiment de *logs*, ja que ara mateix només es pot comprovar la traça dels errors durant l'execució del codi, havent de repetir un error si es vol trobar d'on ve. Es crearà una lògica que permetrà guardar el seguiment d'aquest de forma automàtica en un fitxer.

Llavors, un cop la lògica del sistema de control és vàlida, s'integrarà aquest en un prototip real situat a l'edifici CREA. Es comprovarà si aquest funciona correctament i de forma eficient.

Finalment, es crearà mitjançant containers de Docker imatges dels components més importants del sistema de control per assegurar que en qualsevol dispositiu es pugui executar de forma correcta.

7 Bibliografia

1. Inmocolonial. (s. f.). *Edificis Net Zero: produeixen tota l'energia que consumeixen.* Recuperat de *https://www.inmocolonial.com/ca/blog/net-zero-energy-edificis-produeixen-lenergia-consumeixen*

2. Institut Geogràfic Nacional. (s. f.). *Atlas Nacional d'Espanya: Energia.* Recuperat de https://atlasnacional.ign.es/wane/Energ%C3%ADa

3. Zhou, Y., Zhang, Y., Yang, S., Zhang, K., Xue, C., Ren, H., Yu, Y., & Liu, Y. (2025). Synergetic integration of solar heating and radiative cooling for self-adaptive dual-mode temperature regulation. *Cell Reports Physical Science, 6*(7), 101895. https://doi.org/10.1016/j.xcrp.2024.101895

4. Brita. (s. f.). *¿Qué es una red neuronal artificial (ANN)?* Recuperat de https://brita.mx/que-es-una-red-neuronal-artificial-ann/

5. LinkedIn. (s. f.). *Qué es el overfitting y cómo evitarlo en machine learning.* Recuperat de *https://es.linkedin.com/pulse/qu%C3%A9-es-el-overfitting-y-c%C3%B3mo-evitarlo-en-machine-learning-8a7kf*

6. IBM. (s. f.). *Deep Learning.* Recuperat de https://www.ibm.com/es-es/think/topics/deep-learning

7. Red Hat. (s. f.). *¿Qué son las APIs (interfaces de programación de aplicaciones)?* Recuperat de *https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces*

8. Amazon Web Services. (s. f.). *Diferencias entre SOAP y REST.* Recuperat de https://aws.amazon.com/es/compare/the-difference-between-soap-rest/

9. Sensedia. (s. f.). *Tipos de APIs.* Recuperat de https://www.sensedia.com.es/pillar/tipos-de-apis

10. SD Industrial. (s. f.). *Sistemas de control industrial.* Recuperat de https://sdindustrial.com.mx/blog/sistemas-de-control/

11. RESTfulAPI.net. (s. f.). *Introducción a las APIs REST.* Recuperat de https://restfulapi.net/

12. W3C. (s. f.). *SOAP Version 1.2 Part 1: Messaging Framework.* Recuperat de https://www.w3.org/TR/soap/

13. Pandas. (s. f.). *Documentació oficial: Getting started.* Recuperat de https://pandas.pydata.org/docs/getting_started/index.html

14. Keras. (s. f.). *ReduceLROnPlateau Callback*. Recuperat de https://keras.io/api/callbacks/reduce_lr_on_plateau/

15. Joblib. (s. f.). *Serialización con joblib.dump*. Recuperat de https://joblib.readthedocs.io/en/latest/generated/joblib.dump.html

16. Jia, H., Li, Q., Cheng, Y., et al. (2025). Outlet water temperature prediction of energy pile based on spatial-temporal feature extraction through CNN–LSTM hybrid model. *ScienceDirect*. Recuperat de https://www.sciencedirect.com/science/article/pii/S2666386425002607

17. Li, X., Wang, Y., Chen, Z., et al. (2022). *Machine Learning for Harnessing Thermal Energy*.

18. Zhang, L., Hu, J., Zhao, M., et al. (2023). *CNN–LSTM Hybrid Model for Energy Piles*.

19. Gupta, R., Sharma, A., & Mehta, K. (2021). *Deep Reinforcement Learning for Smart Building Heating Control*.

20. Postman, Postman API Platform. [En línia]. Recuperat de: https://www.postman.com/

21. Christian López García, "Proyecto RCE - Training," projecte intern, Universitat de Lleida, 2024. Codi font facilitat per correu electrònic.

22. Albert Joan Santacana Gabella, "INTERFÍCIE DE COMUNICACIÓ ENTRE UN MODEL DE XARXA NEURONAL LÒGIC PREDICTIU I UN PLC EN UN PROTOTIPUS REAL," Treball de Fi de Grau, Universitat de Lleida, 2023.

A Codi font Sistema RCE - Training

En aquesta secció s'explicaran els fitxers que es van aconseguir del treball previ *Proyecto RCE - Training* i els canvis que hi van haver.

A.1 Fitxers reutilitzats

Els documents que es van reutilitzar i modificar van ser els següents:

- **main.py**: El fitxer *main.py* és el programa principal. Aquest es el que fa les crides a les prediccions. Aquest document no ha estat modificat, ja que la lògica ja era una correcta, amb un bon tractament d'errors i la crida de les prediccions ja estava programada de forma escalable.
- **demand_predictor.py**: El programa que calcula la demanda futura. Actualment encara no s'ha canviat ja que les proves s'han fet en un entorn de test, però en un futur es canviarà.
- **tempereature_predictor.py**: El programa que calcula la temperatura dels dos tancs. S'ha canviat el model de IA que estava per un que funcionés i s'ha millorar la forma de fer prediccions.
- **production_predictor.py**: El fitxer que calcula la producció de la màquina. No s'ha canviat la lògica del document al ser eficient com ja estava.
- **rain_predictor.py**: El fitxer que calcula l'index de pluja en tot el dia. S'ha cambiat la API externa ja que la que estava previament, AEMET, donava errors de connexió de forma constant i es va canviar per Meteo-Sat.

Aquest codi s'ha utilitzat com la base del codi font de la API, i proporciona els quatre tipus de prediccions al Sisitema de Control.