Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Saulės elektrinių generuojamos galios prognozavimo modelių kūrimas ir integracija į elektrinės valdymo sistemą

Baigiamasis magistro projektas

Domas Senkus

Projekto autorius

Asist. Dr. Vygandas Vaitkus

Vadovas

Kaunas, 2026

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Saulės elektrinių generuojamos galios prognozavimo modelių kūrimas ir integracija į elektrinės valdymo sistemą

Baigiamasis magistro projektas

Valdymo technologijos (6211EX014)

Domas Senkus

Projekto autorius

Asist. Dr. Vygandas Vaitkus

Vadovas

Asist. Dr. Kęstas Rimkus

Recenzentas

Kaunas, 2026

Kauno technologijos universitetas

Elektros ir elektronikos fakultetas

Domas Senkus

Saulės elektrinių generuojamos galios prognozavimo modelių kūrimas ir integracija į elektrinės valdymo sistemą

Akademinio sąžiningumo deklaracija

Patvirtinu, kad:

1. baigiamąjį projektą parengiau savarankiškai ir sąžiningai, nepažeisdama(s) kitų asmenų autoriaus ar kitų teisių, laikydamasi(s) Lietuvos Respublikos autorių teisių ir gretutinių teisių įstatymo nuostatų, Kauno technologijos universiteto (toliau – Universitetas) intelektinės nuosavybės valdymo ir perdavimo nuostatų bei Universiteto akademinės etikos kodekse nustatytų etikos reikalavimų;

2. baigiamajame projekte visi pateikti duomenys ir tyrimų rezultatai yra teisingi ir gauti teisėtai, nei viena šio projekto dalis nėra plagijuota nuo jokių spausdintinių ar elektroninių šaltinių, visos baigiamojo projekto tekste pateiktos citatos ir nuorodos yra nurodytos literatūros sąraše;

3. įstatymų nenumatytų piniginių sumų už baigiamąjį projektą ar jo dalis niekam nesu mokėjęs (-usi);

4. suprantu, kad išaiškėjus nesąžiningumo ar kitų asmenų teisių pažeidimo faktui, man bus taikomos akademinės nuobaudos pagal Universitete galiojančią tvarką ir būsiu pašalinta(s) iš Universiteto, o baigiamasis projektas gali būti pateiktas Akademinės etikos ir procedūrų kontrolieriaus tarnybai nagrinėjant galimą akademinės etikos pažeidimą.

Domas Senkus

Patvirtinta elektroniniu būdu

Senkus, Domas. Saulės elektrinių generuojamos galios prognozavimo modelių kūrimas ir integracija į elektrinės valdymo sistemą. Magistro baigiamasis projektas / vadovas Asist. Dr. Vygandas Vaitkus; Kauno technologijos universitetas, Elektros ir elektronikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): elektronikos inžinerija, inžinerijos mokslai .

Reikšminiai žodžiai: saulės elektrinė, fotovoltinė elektrinė, generuojamos galios prognozavimas, aktyvioji galia, MLP neuroninis tinklas, mašininis mokymasis, PLC valdiklis, CODESYS, MQTT, Docker, InfluxDB, modelio integracija, realaus laiko valdymas.

Kaunas, 2026. 80 p.

Santrauka

Šiame baigiamajame magistro projekte nagrinėjamas saulės elektrinių generuojamos aktyviosios galios prognozavimo modelių kūrimas ir jų integracija į pramoninę elektrinės valdymo sistemą. Darbo aktualumas siejamas su didėjančia atsinaujinančios energetikos dalimi elektros energetikos sistemoje ir poreikiu realiuoju laiku tiksliai įvertinti fotovoltinės elektrinės maksimaliai galimą generuoti galią. Ši reikšmė svarbi elektrinės galios ribojimui, rezervo palaikymui, tinklo operatorių reikalavimų vykdymui ir patikimam elektrinės valdymui.

Darbe atlikta saulės elektrinių generuojamos galios prognozavimo modelių apžvalga, aptariant fizikinius, analitinius, statistinius, mašininio mokymosi ir hibridinius metodus. Tiriamojoje dalyje sukurti ir palyginti keli prognozavimo modeliai: MLP, RF, ARIMA, RNN, LSTM, CNN ir hibridinės struktūros. Modeliai vertinti pagal MAE, RMSE ir MSE paklaidas, taip pat grafiškai lyginant jų prognozuojamos galios kreives su realia aktyviąja galia ir šiuo metu elektrinėje naudojamu Laplaso transformacijos pagrindu veikiančiu modeliu.

Tyrimo metu nustatyta, kad nors kai kuriems modeliams, tam tikromis sąlygomis, pasiekus gerą tikslumą, jų realizacija PLC valdiklyje buvo sudėtingesnė dėl didelio parametrų kiekio, būsenų saugojimo arba sekų apdorojimo poreikio. MLP modelis pasirinktas kaip geriausias kompromisas tarp tikslumo, parametrų kiekio, skaičiavimo paprastumo ir realizavimo CODESYS ST aplinkoje.

Svarbus darbo rezultatas buvo parengta modelio parametrų eksportavimo, perdavimo ir taikymo realioje valdymo sistemoje IoT infrastruktūra. Modelio parametrai perduodami į PLC per MQTT sąsają, o valdiklyje dekoduojami ir naudojami aktyviosios galios prognozei realiuoju laiku. Taip pat sukurta konteinerizuota automatinė modelio mokymo sistema, veikianti Linux VM aplinkoje, kuri automatiškai surenka duomenis iš laiko eilučių duomenų bazės, paruošia mokymo imtį, filtruoja ribotos generacijos būsenas, apmoko modelius ir atrenka geriausią variantą. Galutinė MLP sistema testavimo imtyje pasiekė RMSE = 540,331 kW ir MAE = 157,201 kW. Realių elektrinės darbo dienų bandymai parodė, kad PLC valdiklyje veikiantis MLP modelis gali pasiekti mažesnes paklaidas nei šiuo metu taikomas Laplaso transformacijos modelis. MLP modelio MAE paklaida buvo mažesnė 41,50 %, o RMSE – 35,37 %. Tai patvirtina, kad sukurtas modelis ir jo integravimo infrastruktūra gali būti taikomi realioje fotovoltinės elektrinės valdymo sistemoje.

Senkus, Domas. Development of solar power forecasting models and their integration into an industrial power plant control system. Master's Final Degree Project / supervisor Asist. Dr. Vygandas Vaitkus; Faculty of Electrical and Electronics, Kaunas University of Technology.

Study field and area (study field group): electronics engineering, engineering sciences.

Keywords: solar power plant, photovoltaic power plant, power forecasting, active power, MLP neural network, machine learning, PLC controller, CODESYS, MQTT, Docker, InfluxDB, model integration, real-time control.

Kaunas, 2026. 80 p.

Summary

This master's thesis examines the development of photovoltaic power generation forecasting models and their integration into an industrial power plant control system. The relevance of the work is associated with the increasing share of renewable energy in the electrical power system and the need to accurately estimate, in real time, the maximum available power of a photovoltaic power plant. This value is essential for power curtailment, reserve maintenance, compliance with grid operator requirements, and reliable plant control.

The thesis presents a review of photovoltaic power forecasting models, covering physical, analytical, statistical, machine learning, and hybrid methods. In the experimental part, several forecasting models were developed and compared: MLP, RF, ARIMA, RNN, LSTM, CNN, and hybrid structures. The models were evaluated using MAE, RMSE, and MSE metrics, as well as by graphical comparison of predicted power curves with actual active power and the currently used Laplace transform-based model in the power plant.

The study found that although some models achieved high accuracy under certain conditions, their implementation in a PLC controller was more complex due to a large number of parameters, state storage requirements, or sequence processing needs. The MLP model was selected as the best compromise between accuracy, number of parameters, computational simplicity, and feasibility of implementation in the CODESYS ST environment.

An important outcome of the work is the developed IoT infrastructure for exporting, transmitting, and applying model parameters in a real control system. Model parameters are transmitted to the PLC via an MQTT interface, where they are decoded and used for real-time active power prediction. In addition, a containerized automated model training system was developed, operating in a Linux VM environment. This system automatically collects data from a time-series database, prepares training datasets, filters curtailed generation states, trains models, and selects the best-performing one. The final MLP system achieved RMSE = 540.331 kW and MAE = 157.201 kW on the test dataset. Real power plant operation tests demonstrated that the MLP model running on the PLC controller can achieve lower errors than the currently used Laplace transform-based model. The MLP model reduced MAE by 41.50% and RMSE by 35.37%. These results confirm that the developed model and its integration infrastructure can be successfully applied in a real photovoltaic power plant control system.

Turinys

Lentelių sąrašas

Paveikslų sąrašas

Santrumpų ir terminų sąrašas

Santrumpos:

IoT – daiktų internetas *(angl. Internet of Things)*

PLC – programuojamas loginis valdiklis *(angl. Programmable Logic Controller)*

PPC – elektrinės valdiklis *(angl. Power Plant Controller)*

ST – struktūrinis tekstas *(angl. Structured Text)*

PV – fotovoltinis, fotovoltinė elektrinė *(angl. Photovoltaic)*

TSO – perdavimo sistemos operatorius *(angl. Transmission System Operator)*

DSO – skirstymo sistemos operatorius *(angl. Distribution System Operator)*

OPC UA – atviro platforminio ryšio vieningoji architektūra *(angl. Open Platform Communications Unified Architecture)*

MQTT – lengvas pranešimų perdavimo protokolas *(angl. Message Queuing Telemetry Transport)*

SCADA – priežiūros, valdymo ir duomenų surinkimo sistema *(angl. Supervisory Control and Data Acquisition)*

MPP – maksimalios galios taškas *(angl. Maximum Power Point)*

MPPT – maksimalios galios taško sekimas *(angl. Maximum Power Point Tracking)*

aFRR – automatinis dažnio atkūrimo rezervas *(angl. automatic Frequency Restoration Reserve)*

mFRR – rankinis dažnio atkūrimo rezervas *(angl. manual Frequency Restoration Reserve)*

POD – galios svyravimų slopinimo funkcija *(angl. Power Oscillation Damping)*

CPU – centrinis procesorius *(angl. Central Processing Unit)*

P – aktyvioji galia *(angl. active power)*

Q – reaktyvioji galia *(angl. reactive power)*

Pmax – maksimali galia *(angl. Maximum Power)*

BESS – baterijų energijos kaupimo sistema *(angl. Battery Energy Storage System)*

SQL – struktūruota užklausų kalba *(angl. Structured Query Language)*

HTTP – hiperteksto perdavimo protokolas *(angl. Hypertext Transfer Protocol)*

API – programų sąsaja *(angl. Application Programming Interface)*

ML – mašininis mokymasis *(angl. Machine Learning)*

DI – dirbtinis intelektas *(angl. Artificial Intelligence)*

DNT – dirbtinis neuroninis tinklas

RNN – rekursinis neuroninis tinklas *(angl. Recurrent Neural Network)*

LSTM – ilgalaikės ir trumpalaikės atminties tinklas *(angl. Long Short-Term Memory)*

CNN – konvoliucinis neuroninis tinklas *(angl. Convolutional Neural Network)*

GRU – vartų rekursinis vienetas *(angl. Gated Recurrent Unit)*

MLP – daugiasluoksnis perceptronas *(angl. Multilayer Perceptron)*

Adam – adaptyvus momentų įvertinimo optimizavimo algoritmas *(angl. Adaptive Moment Estimation)*

ReLU – tiesinė vienkryptė aktyvacijos funkcija *(angl. Rectified Linear Unit)*

RF – atsitiktinių miškų metodas *(angl. Random Forest)*

RMSE – šakninė vidutinė kvadratinė paklaida *(angl. Root Mean Square Error)*

MAPE – vidutinė absoliutinė procentinė paklaida *(angl. Mean Absolute Percentage Error)*

nMAE – normalizuota vidutinė absoliutinė paklaida *(angl. normalized Mean Absolute Error)*

R^2 – determinacijos koeficientas *(angl. coefficient of determination)*

ARIMA – autoregresinis integruotas slankiojo vidurkio modelis *(angl. AutoRegressive Integrated Moving Average)*

BRB – tikėjimo taisyklių bazė *(angl. Belief Rule Base)*

CAISO – Kalifornijos nepriklausomas sistemos operatorius *(angl. California Independent System Operator)*

ENTSO-E – Europos elektros perdavimo sistemos operatorių tinklas *(angl. European Network of Transmission System Operators for Electricity)*

NREL – Nacionalinė atsinaujinančios energetikos laboratorija *(angl. National Renewable Energy Laboratory)*

DC/DC – nuolatinės srovės keitiklis iš nuolatinės srovės į nuolatinę srovę *(angl. Direct Current to Direct Current converter)*

VM – virtualioji mašina *(angl. Virtual Machine)*

ENV – aplinkos kintamieji *(angl. Environment Variables)*

Terminai:

Dockerizacija / konteinerizacija – procesas, kai programa pritaikoma veikti naudojant *„Docker“* konteinerius. Tai reiškia, kad aplikacija, jos priklausomybės ir aplinka supakuojamos į vieną vienetą (konteinerį), kurį galima lengvai perkelti ir paleisti bet kur.

Nedageneracija – elektrinės veikimo būsena, kai faktinė generuojama galia yra mažesnė už PPC ar kitos valdymo sistemos nustatytą tikslinę galią.

Telematavimas – nuotolinis matavimo duomenų surinkimas ir perdavimas iš įrenginių (pvz., elektrinės) į valdymo ar stebėjimo sistemą realiu laiku.

Pilkoji dėžė (angl. *grey-box*) – metodas ar sistema, kurioje dalis veikimo yra žinoma (pvz., fizikiniai principai), o dalis nustatoma iš duomenų ar laikoma nežinoma. Tai tarpinis variantas tarp visiškai žinomos ir visiškai nežinomos sistemos.

Kraštinė kompiuterija (angl. *edge*) – skaičiavimų atlikimas arti duomenų šaltinio (pvz., elektrinėje ar PLC), o ne centralizuotame serveryje ar debesijoje. Tai leidžia sumažinti vėlavimą ir padidinti sistemos patikimumą realiu laiku.

Robustiškumas – modelio gebėjimas tiksliai veikti net esant duomenų triukšmui, matavimo paklaidoms ar staigiems apšvietos pokyčiams.

Topologija – sistemos elementų išdėstymo ir tarpusavio sujungimo struktūra.

Laplaso transformacija – matematinis metodas, leidžiantis laiko srities signalus perkelti į kompleksinę sritį, kur patogiau analizuoti sistemas (pvz., perdavimo funkcijas).

Pyranometras – prietaisas, matuojantis saulės spinduliuotės intensyvumą (W/m^2).

Konvoliucija – operacija, nusakanti dviejų signalų tarpusavio sąveiką, naudojama signalų apdorojime ir dirbtiniuose neuroniniuose tinkluose.

Versijavimas – skirtingų kodo ar modelio versijų valdymas ir sekimas.

Metodologija – tyrimo metodų ir jų taikymo principų visuma.

Validavimas – modelio ar sistemos tikrinimas, siekiant įvertinti jo atitikimą realiems duomenims.

Iteratyvumas – pakartotinis sprendimo tobulinimas per kelis nuoseklius ciklus.

Azimutas – krypties kampas horizontalioje plokštumoje, matuojamas laipsniais nuo šiaurės.

Normalizacija – duomenų perkėlimas į bendrą skalę, siekiant pagerinti modelio veikimą.

Denormalizacija – normalizuotų duomenų grąžinimas į pradinę skalę.

Interferencija – signalų tarpusavio sąveika, dėl kurios jie gali sustiprėti arba susilpnėti.

Sekiklis – įrenginys, automatiškai orientuojantis saulės modulius į saulę.

***Few-shot* mokymasis** – mašininio mokymosi metodas, kai modelis išmoksta atlikti užduotį naudodamas labai nedidelį kiekį mokymo duomenų.

Modbus – pramoninis protokolas, skirtas duomenų apsikeitimui tarp įrenginių.

CODESYS – valdiklių programavimo sistema

Python – programavimo kalba

Įvadas

Didelės galios fotovoltinėse elektrinėse vienas svarbiausių valdymo uždavinių yra tikslus momentinės maksimaliai galimos generuoti aktyviosios galios įvertinimas. Ši reikšmė reikalinga ne tik elektrinės darbui stebėti, bet ir galios ribojimui, rezervo palaikymui, tinklo paslaugoms bei atitikties bandymams. Praktikoje dažnai naudojami fizikiniai arba analitiniai modeliai, paremti apšvietos ir temperatūros matavimais, tačiau reali elektrinės elgsena yra sudėtingesnė. Ją veikia debesuotumas, modulių temperatūra, keitiklių darbo režimai, elektrinės topologija, valdymo ribojimai, matavimų vėlavimai ir duomenų kokybė. Dėl šių priežasčių įprastiniai modeliai ne visada tiksliai atspindi realų elektrinės galios potencialą.

Šiame baigiamajame magistro projekte nagrinėjamas saulės elektrinės generuojamos galios prognozavimo modelių kūrimas ir jų integracija į elektrinės valdymo sistemą. Darbe analizuojami skirtingi prognozavimo modeliai, vertinamas jų tikslumas, praktinis pritaikomumas PLC aplinkoje ir galimybė automatizuotai atnaujinti modelio parametrus. Darbo praktinis aktualumas siejamas su realios fotovoltinės elektrinės eksploatacija. Tyrime naudojami istoriniai elektrinės generacijos, apšvietos, temperatūros ir laiko duomenys, o sukurti modeliai lyginami su realia aktyviąja galia bei šiuo metu taikomu Laplaso transformacijos pagrindu veikiančiu modeliu. Galutinis darbo rezultatas yra ne tik prognozavimo modelis, bet ir IoT programinė infrastruktūra, leidžianti automatiškai surinkti duomenis, apmokyti modelį, palyginti jį su ankstesnėmis versijomis ir perduoti parametrus į PLC valdiklį.

Darbo tikslas – sukurti ir eksperimentiškai įvertinti saulės elektrinės generuojamos galios prognozavimo modelį bei jo integravimo į pramoninę elektrinės valdymo sistemą sprendimą, užtikrinantį prognozavimą realiuoju laiku.

Darbo uždaviniai:

1. atlikti saulės elektrinių generuojamos galios prognozavimo modelių apžvalgą.

2. sukurti prognozavimo modelius, naudojant istorinius fotovoltinės elektrinės generacijos bei meteorologinius duomenis, ir optimizuoti jų parametrus siekiant didžiausio tikslumo.

3. suprojektuoti ir realizuoti konteinerizuotą programinę infrastruktūrą, kuri valdytų automatinį duomenų surinkimą ir modelių permokymą

4. parengti ir įgyvendinti duomenų perdavimo sprendimą modelių integracijai į pramoninį valdiklį, užtikrinant prognozių pateikimą realiuoju laiku.

5. eksperimentiškai įvertinti sukurtos sistemos efektyvumą, lyginant skirtingų modelių tikslumą (RMSE, MAE ar kitais kokybės įvertinimo rodikliais) su realiais elektrinės generacijos duomenimis bei šiuo metu taikomais generuojamos galios prognozavimo sprendimais.

1. Literatūrinė apžvalga

Šioje literatūrinėje analizėje nagrinėjami fotovoltinių elektrinių generuojamos galios realaus laiko modeliavimo ir integravimo į elektrinės valdymo sistemą sprendimai. Pirmiausia aptariama, kodėl auganti saulės ir vėjo generacijos dalis kelia naujus tinklo stabilumo, galios rezervo ir valdymo patikimumo reikalavimus. Toliau analizuojama PPC valdiklio reikšmė didelės galios PV elektrinėse ir poreikis tiksliai įvertinti momentinę maksimaliai prieinamą generuojamą galią. Literatūroje lyginami fizikiniai, analitiniai, pilkosios dėžės, mašininio mokymosi ir hibridiniai modeliai, įvertinant jų tikslumą, skaičiavimo sudėtingumą bei tinkamumą realaus laiko taikymui. Taip pat apžvelgiami modelių diegimo į PLC arba decentralizuotas *edge* sistemas principai, įskaitant „Docker“, OPC UA ir MQTT technologijų taikymą. Ši analizė reikalinga siekiant pagrįsti, kokie metodai yra tinkamiausi kuriant praktiškai pritaikomą PV galios prognozavimo modelį, kuris galėtų veikti pramoninėje elektrinės valdymo aplinkoje.

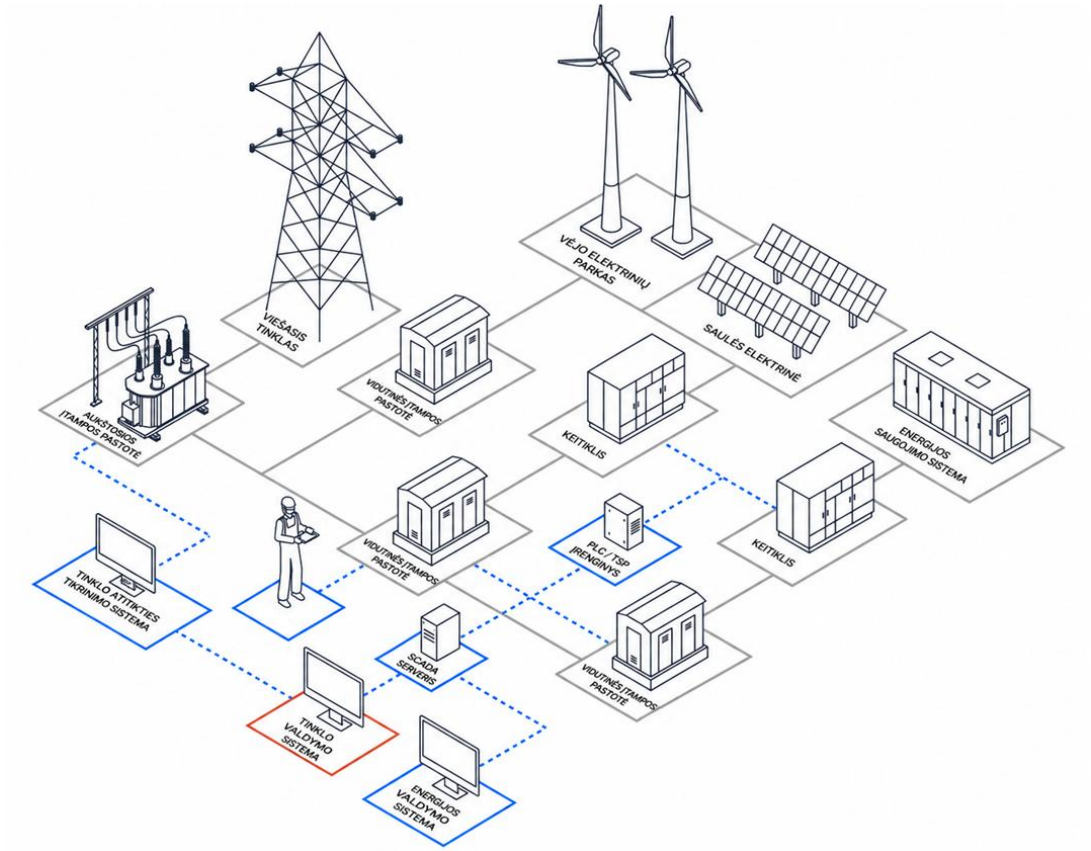
1.1. Atsinaujinančios energetikos plėtra ir tinklo stabilumo iššūkiai

Pastaraisiais dešimtmečiais atsinaujinančios energetikos sektorius patyrė milžinišką augimą. Pramoniniu mastu pradėta naudoti hidroelektrinių energija, vėliau išplėtotos vėjo jėgainės ir saulės PV elektrinės – pastarųjų diegimas šiuo metu yra pasiekęs aukščiausią piką. [1]Didėjant nepastovių saulės ir vėjo išteklių daliai elektros energetikos sistemoje, aktualiu tapo generacijos ir apkrovos disbalanso klausimas. Siekiant subalansuoti gamybą ir vartojimą, pradėti diegti energijos kaupimo sprendimai – tiek cheminės (baterijos), tiek fizinės formos (pvz., skriejikai, suslėgtų dujų saugyklos) kaupikliai. Šie įrenginiai leidžia kaupti perteklinę energiją ir panaudoti ją tada, kai generacija sumažėja, o paklausa išauga. Tačiau net ir su kaupikliais didelis atsinaujinančių išteklių tankis kelia naujų iššūkių perdavimo ir skirstymo tinklų operatoriams, kurie privalo užtikrinti tinklo stabilumą realiu laiku [1]. Elektros energetikos tinklo sistema veikia kaip gyvas organizmas – norint išlaikyti patikimumą, būtina nuolat palaikyti balansą tarp generacijos ir apkrovos bei kontroliuoti tinklo parametrus (dažnį, įtampą, reaktyviąją galią). Augant keitiklinių *(angl. inverter)* šaltinių (saulės, vėjo, baterijų) skaičiui, mažėja sistemos inercija ir tampa sudėtingiau valdyti dažnio bei įtampos svyravimus [2]. Atsinaujinančių išteklių integracija jau lėmė keletą ekstremalių situacijų pasaulyje. Pavyzdžiui, 2025 m. balandį Iberijos pusiasalį apėmęs plataus masto elektros gedimas parodė, kad esant didelei atsinaujinančios generacijos daliai būtina užtikrinti tinkamą įtampos ir dažnio kontrolę – po šio incidento Ispanijos ataskaitose pabrėžta, jog atsinaujinantys šaltiniai nebuvo tiesioginė avarijos priežastis, o ateityje, priešingai, gali tapti sprendimo dalimi, jei jiems bus leidžiama aktyviai stabilizuoti tinklą[3]. Pagal šį pavyzdį galima daryti išvadą, kad norint saugiai pasiekti dideles saulės ir vėjo generacijos apimtis, būtina diegti tinklo stabilizavimo funkcijas tokias kaip dažnio reguliavimas, įtampos kontrolė, sintetinė inercija [1].

1.1.1. Tinklo operatorių reikalavimai didelės galios PV elektrinėms

Perdavimo TSO ir skirstymo DSO tinklų operatoriai, reaguodami į augančią atsinaujinančios generacijos dalį, griežtina prisijungimo sąlygas naujiems elektrinių parkams. Lietuvoje, kaip ir kitose šalyse, galioja elektros tinklų kodeksai, numatantys įvairias elektrinių kategorijas pagal įrengtąją galią (pvz., A, B, C, D tipo elektrinės) ir joms taikomus reikalavimus. Mažesniems įrenginiams (pvz., <250 kW, A tipo) keliami minimalūs techniniai reikalavimai, tačiau vidutinės (250 kW–5 MW, B tipo) ir ypač didelės galios elektrinės (5 - 15 MW, C ir >15MW D tipai) privalo turėti integruotus valdymo sprendimus, leidžiančius operatoriui valdyti jų darbą realiuoju laiku. Nuo maždaug 5 MW

galios slenksčio saulės elektrinės paprastai tampa svarbios perdavimo tinklui, todėl TSO (Lietuvoje – Litgrid) reikalauja tokių elektrinių dalyvavimo pirminiuose dažnio reguliavimo procesuose ir teikti galios rezervą dažnio atkūrimui [4]. Tai praktiškai reiškia, kad PV elektrinė turi dirbti ne visu maksimaliu galingumu, o palaikyti nustatytą procentinį rezervą (pvz., ~5–10 % galios) virš tiekiamos galios, kurį būtų galima panaudoti dažniui pakritus. Tam, kad elektrinė galėtų patikimai užtikrinti tokį galios padidėjimą, būtina realiu laiku žinoti, kokia yra maksimali įmanoma generuoti galia esamomis sąlygomis. Ši didžiausios galios *(angl. available power)* vertė nuolat kinta priklausomai nuo saulės apšvietos *(angl. irraidiance)*, modulių temperatūros, panelių švarumo ir kitų veiksnių. Realaus laiko modelis, leidžiantis apskaičiuoti fotovoltinės elektrinės momentinį maksimalų galios potencialą, tapo esminiu įrankiu siekiant išpildyti tinklo operatoriaus sąlygas [1]. Pvz., Kalifornijos sistemos operatoriaus CAISO taisyklėse numatytos baudos generacijos nuokrypiams, jei elektrinė neįvykdo deklaruoto galios rezervo ar dispečerinio grafiko – todėl būtina tiksliai įvertinti, kiek galios iš tiesų galima generuoti tam tikru momentu, kad nepaaiškėtų, jog rezervas pervertintas [1]. Analogiškai Europos perdavimo tinkluose galioja ENTSO-E rekomendacijos, kad kintamos generacijos šaltiniai teiktų savo indėlį į dažnio ir galios balansavimą [1]. PV elektrinės šiuo požiūriu neatsilieka nuo vėjo – modernios saulės jėgainės aprūpinamos elektrinių valdikliais PPC, kurie gali automatiškai riboti galią, sekti sistemos operatoriaus užduotis bei perduoti telematavimus apie elektrinės būklę.



1 pav. Elektrinės valdymo sprendimo (PPC) architektūrinė schema

Vienas iš svarbiausių PPC funkcijų didelėse saulės elektrinėse yra galios rezervo palaikymas realiuoju laiku. Pvz., jei „Litgrid“ reikalauja 5 % rezervo, elektrinė nuolat palaikys generaciją ties ~95 % galimo maksimumo. Akivaizdu, kad tokiam valdymui PPC turi bet kurią akimirką žinoti, koks tas 100 % būtų, t. y. koks yra momentinis MPP. Be šio žinojimo neįmanoma nei patikimai apriboti

galios, nei garantuoti, kad rezervas realus ir atitinkamas. Todėl PV elektrinės galios modelis realiuoju laiku yra būtinas ir tinklo stabilumui užtikrinti, ir komercinėms operacijoms (balanso grupių valdymui, prekybai elektra). Šis leidžia prognozuoti bei deklaruoti prieinamą generaciją su minimaliu nuokrypiu. Reikalavimai šiam modeliui yra griežti – jis turi būti tikslus ir greitaeigis, kad pateiktų patikimą informaciją net staigių sąlygų pokyčių (pvz., praplaukiant debesims) metu. Be to, modelis turėtų veikti be papildomų vėlavimų, integruotas į elektrinės valdymo sistemą.

1.1.2. PV elektrinės atitikties vertinimas

Lietuvos sąlygomis modelio poreikį sustiprina ir formalus elektrinių atitikties vertinimas realiomis eksploatacijos sąlygomis. Skirstymo operatorius „ESO“ „natūrinius (atitikties) bandymus“ apibrėžia kaip bandymus, kuriais siekiama įrodyti, kad elektros energijos gamybos modulių veikimo charakteristikos atitinka Europos Komisijos tinklo kodekso reikalavimus – „ESO“ nurodo [5], kad bandymai reikalingi visoms elektrinėms, kurių didžiausias pajėgumas *Pmax* daugiau nei 100 kW (taip pat ir kitais teisės aktuose apibrėžtais atvejais), o bandymų proceso pabaigoje parengiama atitikties vertinimo ataskaita kaip galutinis dokumentas, patvirtinantis bandymų atlikimą [5, 6]. Dėl tokios procedūros praktikoje būtina iš anksto suprojektuoti matavimų, telematavimų ir valdymo logiką taip, kad bandymų metu būtų galima pakartojamai įrodyti reikalaujamą elektrinės greitaveiką bei valdymo funkcijų korektiškumą.

Perdavimo tinklo lygmeniu „Litgrid“ nustato, kad prieš pripažįstant tinkamas eksploatuoti elektrines būtina jas išbandyti ir atlikti matavimus pagal reglamento reikalavimus ir gamintojų instrukcijas [7]. Be to, reglamente akcentuojama, kad bandymų metu turi būti surinkti pradiniai duomenys, reikalingi būklės pokyčiams vertinti eksploatacijos metu [7]. PV elektrinių valdymo sistemoje tai tiesiogiai siejasi su galios rezervų ir dažnio reguliavimo funkcijų patikimumu: norint pagrįstai deklaruoti rezervą, PPC turi remtis patikrinta, matavimais paremta realaus laiko galios modelio reikšme. Taigi realaus laiko maksimalios galios modelis tampa ne tik optimizavimo priemone, bet ir atitikties įrodymo bei eksploatacinio patikimumo užtikrinimo dalimi.

1.2. PV galios realaus laiko modeliavimo metodų apžvalga

1.2.1. Fizikiniai ir analitiniai modeliai

Tradiciškai saulės elektrinės galia skaičiuojama naudojant fizikiniais dėsniais paremtus modelius. Dažniausiai remiamasi panelių I–V charakteristikų analize: vieno diodo modeliu ar jo dariniais (pvz., penkių parametrų modeliu), kurie sieja modulio srovę ir įtampą su apšvieta bei temperatūra. Žinant momentinę saulės spinduliuotę (W/m²) ir saulės modulio darbinę temperatūrą, galima apskaičiuoti teorinę modulio maksimalią galią. Šią informaciją integruojant per visą elektrinės modulių masyvą, gaunama visos jėgainės maksimali galia. Paprasčiausiu atveju daroma prielaida, kad galia proporcinga apšvietai, ir taikomas nusistovėjęs galios koeficientas: pavyzdžiui, 1000 W/m² apšvietai atitinka nominali galia, o esant mažesnei apšvietai – tiesiog proporcingai mažesnė galia, koreguota pagal modulio efektyvumą ir temperatūrą. Tokie pirmo artumo modeliai gali būti realizuojami net paprasta perdavimo funkcija, į modelį įtraukiant laiko konstantą, apibūdinančią PV sistemos inerciją (pvz., atsaką į debesies perėjimą). Vis dėlto realios saulės elektrinės elgsena yra kompleksiškesnė: ją veikia modulių degradacija, netolygus apšvietimas, keitiklių darbo taško apribojimai, temperatūriniai efektai. Rankiniu būdu suderinti vieno laipsnio perdavimo funkcijos parametrai dažnai nepajėgia tinkamai atspindėti šių reiškinių dinamikos, todėl realiomis eksploatacijos sąlygomis atsiranda reikšmingos modelio paklaidos. Pavyzdžiui, NREL tyrimuose parodyta situacija, kai elektrinės

valdiklis, remdamasis per optimistiniu paprasto modelio įvertinimu, apskaičiavo, kad galios rezervas dar yra, tačiau realybėje DC/DC keitikliai jau veikė savo maksimumo taške – tai vos per 1 s intervalo grafiką rodančiame pavyzdyje sukėlė ~0,5 MW nuokrypį tarp modeliuotos ir tikros galios (apie 3 % elektrinės galios) [1]. Šis pavyzdys rodo, kad net ir nedidelės modelio paklaidos gali turėti reikšmingą įtaką valdymo sprendimams, todėl fizikiniai modeliai reikalauja labai gero sukalibravimo: būtina turėti tikslius modulių parametrus, jutiklius (pyranometrus, elementų temperatūros matuoklius), o net ir tokiu atveju dalis reiškinių (pvz., nešvarumų įtaka ar dalinis šešėliavimas) lieka neįvertinti.

Nepaisant to, literatūroje yra sukurta patikimų fizikinių metodų, skirtų realaus laiko maksimalios galios taškui įvertinti. Pvz., Hoke pasiūlė eksperimentais patikrintą metodą, kur momentinė apšvieta ir elemento temperatūra naudojamos apskaičiuoti MPP galiai, užtikrinant nustatyto rezervo palaikymą [1]. Kiti autoriai kaip Batzelis sukūrė jutiklių nereikalaujantį algoritmą, kai dviejų pakopų PV sistemoje nuolat šiek tiek moduliuojamas DC/DC keitiklio darbinis taškas ir pagal gaunamus įtampos bei srovės duomenis taikomas kreivės aproksimavimas *(angl. curve fitting)*, leidžiantis apskaičiuoti pilnos kreivės MPP esant apribotai galiai [1]. Šis metodas realiu laiku identifikuoja kritinius parametrus, todėl labai tiksliai nustato, kur būtų galios maksimumas, net kai keitiklis veikia daline apkrova [4]. Batzelio darbe pabrėžiama, kad tai pirmas metodas, naudojantis visą fizikinį PV modelį apriboto darbo režimu, ir eksperimentiškai parodo, jog šis metodas pasižymi labai geru tikslumu ir robustiškumu net ir greitai kintančių sąlygų atvejais [4]. Tai rodo, kad tinkamai parengti analitiniai modeliai gali pasiekti aukštą tikslumą. Vis dėlto, jų diegimas gali būti apsunkintas: tokie algoritmai dažnai būna priklausomi nuo keitiklio topologijos ar reikalauja specifinių valdiklio modifikacijų [1]. Pavyzdžiui, vienpakopėse sistemose (kai MPPT vykdomas pačiame keitiklyje) rezervui palaikyti keičiami keitiklio MPPT valdikliai, o dvipakopėse – tenka keisti DC/DC konverterio algoritmą [1]. Kiekvienu atveju tai technologiškai įmanoma, tačiau universalaus sprendimo nėra – modelį reikia pritaikyti konkrečiai įrangai.

Apibendrinant, fizikiniai PV galios modeliai yra tikslūs ir suteikia potencialiai gerą aproksimaciją už laboratorijos ribų, tačiau norint išlaikyti mažą paklaidą, juos būtina periodiškai kalibruoti pagal realius duomenis. Dėl to vis daugiau dėmesio skiriama duomenimis grįstiems modeliams, galintiems automatiškai „išmokti“ elektrinės elgseną, ypač kai elektrinės yra sudarytos iš daugybės topologiškai skirtingų keitiklių.

1.2.2. Analitiniai pilkosios dėžės elektrinės lygmens modeliai

Tarp grynai fizikinio ir grynai duomenimis grįsto ML požiūrio plačiai taikomi analitiniai arba pilkosios dėžės sprendimai, kai elektrinės maksimalios galios priklausomybė nuo aplinkos sąlygų aprašoma supaprastintu dinaminiu modeliu. Šio tipo metodų tikslas – sukurti pakankamai tikslų ir lengvą apskaičiuoti modelį realiuoju laiku, kad jį būtų galima vykdyti PPC/PLC cikle. Dažniausias principas: sudaryti statinę elektrinės charakteristiką ir ją papildyti dinamika, aprašančia elektrinės atsaką į staigius apšvietos ar temperatūros pokyčius.

Dinamikai aprašyti taikomos perdavimo funkcijos Laplaso srityje, pvz., pirmos eilės modelis:

$$\frac{P\_\max(s)}{G(s)} = \frac{Ke^{-\tau d^S}}{Ts+1}; \qquad (1)$$

Čia K – stiprinimo koeficientas (W per W/m²), T – laiko konstanta, o τ_d – grynasis uždelsimas, priklausomai nuo matavimo, komunikacijos ir valdiklio ciklo. Tokia struktūra leidžia aprašyti realios elektrinės inerciją bei kontroliuoti modelio „greitaveiką“ parenkant T. Praktikoje parametrai

identifikuojami iš matavimų: naudojami apšvietos šuoliai (ar natūralūs, ar iš duomenų išskirti greiti pokyčiai), o K, T, τ_d įvertinami mažiausių kvadratų ar maksimaliojo tikėtinumo metodais. Analitinio modelio privalumas – aiškiai apibrėžta dinamika ir galimybė atlikti stabilumo bei jautrumo analizę (pvz., mažų signalų modeliavimą), įtraukiant apšvietos ir elemento temperatūros variaciją kaip darbo taško pokytį [8, 9].

Realioms elektrinėms analitiniai modeliai dažnai kuriami kaip agreguoti aprašai, kad būtų išvengta detalios keitiklių ir DC pusės dinamikos imitavimo, tačiau išlaikoma pakankama tikslumo klasė PPC uždaviniams. Tokie supaprastintos eilės *(angl. reduced-order)* modeliai naudojami tiek laiko srities didelių signalų imitacijoms, tiek mažų signalų analizei, ypač kai aktualios silpno tinklo sąlygos ir žemo dažnio virpesiai [10]. Nors šie darbai dažnai orientuoti analizuoti stabilumą, jų metodologija tiesiogiai perkeliama į *P_available* realaus laiko įverčio kūrimą, kai modeliuojama galia laikoma valdymo grandinės etalonu rezervui palaikyti.

Kai turimų jutiklių skiriamoji geba ribota (pvz., keli pyranometrai dideliame parke), tik minėti matavimai gali sudaryti netikslų modelio veikimą. Parką veikia nevienodas debesuotumas, moduliai skirtingai orientuoti, skirtumai tarp temperatūrų ir pan. Tokiais atvejais taikomos sprendimų lentelės *(angl. decision tables)* metodai, kur lentelė gali būti sudaryta iš istorinių duomenų, tai reiškia, kad K, T, τ_d gali kisti priklausomai nuo taisyklių, kurios „prispaudžia“ modelį prie realybės. Paprastai tai reiškia: skirtingiems darbo taškams taikomi skirtingi linijiniai modeliai ir atitinkamai perjungiami valdiklio parametrai [11]. Tai leidžia turėti kelių režimų analitinį modelį, kuris geriau aprašo netiesiškumus nei vienas globalus K, T modelis.

Sprendimų lentelės gali būti tarsi tarpinis žingsnis iki mašininio mokymosi: jos leidžia paruošti analitinius modelius ir kartu naudoti duomenis sisteminėms paklaidoms kompensuoti. Tai artima taisyklėmis grįstiems modeliams *(angl. rule-based),* kuriuose įvestys susiejamos su išvestimi per taisyklių rinkinį. Literatūroje pasiūlyti ir pažangesni BRB metodai, kurie leidžia optimizuoti taisyklių svorius nepažeidžiant tikslumo [12]. Toks „lentelių + dinamikos“ derinys praktikoje dažnai yra patrauklus PLC/PPC integracijai: skaičiavimai lengvi, elgsena prognozuojama, o tikslumą galima gerinti periodiškai atnaujinant lentelę pagal surinktus parko duomenis.

1.2.3. Dirbtinio intelekto ir duomenimis grįsti modeliai

Dirbtinio intelekto (DI) metodai, ypač mašininio mokymosi ir neuroninių tinklų algoritmai, pastaraisiais metais sparčiai skinasi kelią ir į saulės generacijos modeliavimą. Dar 1997 m. Hiyama ir Kitabayashi pristatė idėją naudoti dirbtinį neuroninį tinklą PV modulių maksimalaus galingumo nustatymui pagal aplinkos informaciją [1]. Tai buvo ankstyvas pavyzdys, kaip neuronai gali išmokti nelinijinius ryšius tarp orų sąlygų ir generuojamos galios. Vėlesni tyrimai plėtojo šią kryptį: jau daugiau kaip dešimtmetį fotovoltinės generacijos trumpalaikės prognozės kuriamos taikant įvairias ML technikas – nuo klasikinių dirbtinių neuroninių tinklų DNT ir atraminių vektorių regresijos iki modernių giliųjų architektūrų, tokių kaip RNN, LSTM, konvoliuciniai tinklai CNN [13, 14]. Mašininio mokymosi privalumas yra gebėjimas fiksuoti sudėtingus, netiesinius priklausomumus tarp daugelio veiksnių (spinduliuotės, oro temperatūros, drėgmės, debesuotumo, laiko ir t.t.) ir gautos galios – tai leidžia ženkliai sumažinti prognozės klaidą, lyginant su tradiciniais empiriniais metodais [14]. Pavyzdžiui, literatūroje nurodoma, kad pažangūs giliojo mokymosi modeliai (LSTM, CNN ir pan.) gali pastebimai aplenkti fizikiniais modeliais grįstas prognozes, ypač trumpalaikiame periode, sumažindami paklaidą keliais procentais [14].

Realiam laikui skirtuose modeliuose DI metodai gali būti taikomi dviem būdais: (1) tiesiogiai nustatyti esamą galios santykį pagal einamąsias aplinkos reikšmes (t. y. iš esmės aproksimuoti sudėtingą fizikinę funkciją neuroniniu tinklu), arba (2) atlikti labai trumpo laikotarpio prognozę (pvz., kelių minučių), kuri esant tinkamam greičiui praktiškai atspindi dabartinę galią. Pirmuoju atveju modelis mokomas su istoriniais duomenimis, kuriuose įvestys – tuo metu fiksuotos sąlygos, o išvestis – faktinė elektrinės generuota galia. Išmokęs tinklas geba apibendrinti neregėtoms situacijoms. Tyrimai rodo, kad gerai parinktas tinklo tipas gali net pranokti fizikinį modelį: pavyzdžiui, 2025 m. atliktame eksperimentiniame palyginime tarp matematinio PV modelio ir trijų neuroninių tinklų (MLP, LSTM, GRU) nustatyta, jog MLP pasiekė didžiausią tikslumą ir stabiliausius rezultatus ištisus metus [15]. MLP pranoko tiek sudėtingesnes giliojo mokymosi architektūras (LSTM, GRU), tiek tradicinę analitinę formulę, ypač tais atvejais, kai ryšiai tarp meteorologinių kintamųjų ir galios buvo gana statiški [15]. Fizikinis modelis konkuravo tik mažo kintamumo sąlygomis (vasarą, esant tolydžiai saulei) [15]. Šie rezultatai atskleidžia, jog vieno universalaus geriausio modelio nėra – modelio tinkamumas priklauso nuo eksploatacijos scenarijaus, sezoniškumo ir duomenų struktūros. Dėl to šiandien neretai pasirenkami hibridiniai metodai arba modelių deriniai, siekiant išnaudoti skirtingų metodų privalumus [14].

1.2.4. Hibridiniai metodai ir modelio adaptavimas

Atsižvelgiant į tai, kad fizikiniai ir duomenimis grįsti metodai turi savų stiprybių, atsirado hibridinių modelių, jungiančių geriausias šių modelių savybes. Vienas būdas – naudoti fizikinio modelio rezultatą kaip papildomą požymį ML modeliui arba koreguoti fizikinį modelį naudojant mašininį mokymąsi. Pavyzdžiui, Santos tyrime tiesinis regresijos modelis buvo apmokytas skirtumo tarp meteorologinės tarnybos prognozės ir faktinių duomenų, kas leido net ir su labai ribotu duomenų kiekiu (naujai elektrinei) patikslinti fizikinio modelio išvestį ir sumažinti prognozės šakninį vidutinį kvadratinį nuokrypį RMSE ~3,7 % [14]. Toks metodas veiksmingas naujai paleidžiamiems parkams, kur istorinės informacijos dar nedaug – iš pradžių pasikliaujama inžineriniu modeliu, tačiau mašina „mokosi“ iš pirmųjų eksploatacijos mėnesių duomenų ir pamažu perima vairą, ištaisydama sistemines paklaidas. Kituose darbuose siūloma naudoti du modelius lygiagrečiai: vieną paprastą ir patikimą nuo pat pradžių (pvz., fizikinį arba perspektyvų kitų jėgainių duomenimis ištreniruotą modelį), ir antrą – besimokantį modelį, kuris startuoja nuo dalinai perimtų žinių, pasitelkiant perkeliamą mokymąsi *(angl. transfer learning)*, ir tobulina tikslumą kaupdamas vietinius duomenis. Laikui bėgant antrasis modelis tampa vis tikslesnis, todėl jo svoris sprendimų priėmime didinamas, o pirmojo – proporcingai mažinamas. Toks dinaminis modelių svėrimo principas leistų garantuoti patikimumą pradiniu etapu ir optimalų tikslumą ilgainiui.

Perkeliamo mokymosi ir nuolatinio *(angl. online)* mokymosi metodai ypač aktualūs fotovoltinių jėgainių modeliavime, kadangi kiekviena elektrinė turi savas ypatybes (skiriasi geografinė vieta, moduliai, keitikliai, aplinkos sąlygos). Greito perkėlimo mokymasis leidžia perimti sukauptas žinias iš kitų panašių PV sistemų arba iš fizikinio modelio, ir pritaikyti jas naujai sistemai, iš esmės „įsibėgėti“ modeliui per ženkliai trumpesnį laiką nei mokantis nuo nulio [13]. Mokslinėje literatūroje aprašomi įvairūs šio principo įgyvendinimo būdai, pavyzdžiui, *few-shot* mokymas, kai modelio korekcijai pakanka nedidelio naujų duomenų kiekio. Kitas svarbus aspektas – nuolatinis modelio prisitaikymas prie kintančių sąlygų. Tam taikomi periodinio persimokymo arba nuolatinio mokymosi metodai, leidžiantys modeliui palaipsniui prisitaikyti prie pokyčių, tokių kaip modulių degradacija, sezoniniai apšvietos skirtumai ar eksploatacijos sąlygų kaita [16]. Tinkamai suprojektuotas adaptyvus modelis ilgainiui gali tapti tikslesnis už bet kokį statiškai nustatytą sprendimą. Tačiau svarbu

pažymėti, kad modelio adaptacija turi vykti kontroliuojamai. Nuolatinio mokymosi mechanizmai paprastai riboja mokymosi žingsnį, saugo nuo pertreniravimo ir užtikrina, kad modelio ar valdiklio parametrai nekistų staiga. Dėl to PV elektrinės galios modelis gali išlaikyti pakankamą tikslumą per visą elektrinės eksploatacijos laikotarpį, net jei pradinėse fazėse paklaidos buvo didesnės.

Modelių efektyvumo vertinimas paprastai atliekamas statistinėmis metrikomis, lyginant modelio spėjimus su realiai išmatuota galia. Pavyzdžiui, Batzelio metodo bandymuose 2 kW modulių stende gauti MPP įvertinimai turėjo paklaidas mažesnes nei 1 %, esant pastovioms sąlygoms, ir vos kelis procentus greitai kintant apšvietai [4]. Kituose tyrimuose dirbtinio intelekto modeliai pasiekė vos ~5 % vidutinės absoliutinės procentinės paklaidos MAPE prognozuojant valandos generaciją naujai elektrinei [2]. Vis dėlto vien vidutinių metrikų nepakanka – tinklo operatoriui svarbu žinoti, kokia gali būti didžiausia momentinė paklaida. Todėl literatūroje analizuojami ir paklaidų pasiskirstymai bei kraštutiniai nuokrypiai. NREL mokslininkų darbe parodyta, kad naudojant tik vieną jutiklį ar referencinį keitiklį, esant intensyviam debesuotumui, galima momentinę galią pervertinti >50 %, tačiau praplėtus sistemą su daugiau referencinių taškų (48 mazgus skirtingose parko dalyse) didžiausia momentinė paklaida sumažėjo iki ~7,9 % [1]. Šis pavyzdys parodo, kad didesnė modelio skiriamoji geba ir geresnis duomenų reprezentatyvumas leidžia reikšmingai sumažinti kraštutines paklaidas. Apibendrinant, modelio sėkmė vertinama pagal tai, ar jam pavyksta nuosekliai išlaikyti žemą paklaidą visuose situacijose: ir saulėtą dieną, ir su staigiais apšvietos pasikeitimais. Geriausi rezultatai pasiekiami derinant fizikos ir duomenų mokymosi privalumus, o taip pat pasitelkiant hibridinius bei prognozinius modelius [14].

1.3. Modelių integravimas automatikos ir valdymo sistemose

1.3.1. Modelio realizavimas elektrinės valdiklyje (PLC)

Norint teorinį modelį paversti praktine funkcija veikiančioje elektrinėje, reikia jį integruoti į automatikos sistemą. Daugumos didesnių PV elektrinių valdymo branduolys yra PLC arba pramoninis kompiuteris, kuriame įdiegta elektrinės valdymo programa (dažnai naudojant IEC 61131-3 standartus, pvz., *CODESYS* aplinką). Ši programa atlieka tokias užduotis kaip generatorių keitiklių valdymas, reikšmių nuskaitymas iš jutiklių (apšvietos, temperatūros), ryšys su tinklo operatoriaus dispečerine ir pan. PV galios modelio funkcija turi būti įterpta į šį valdymo algoritmų rinkinį. Paprastesni fizikiniai modeliai gali būti nesunkiai užkoduojami tiesiogiai PLC logikoje, realizuojant formulę $P\_available = f(G,T)$ su keletu operacijų. Tuo tarpu sudėtingesni DI modeliai tradiciškai buvo laikomi per sunkiais PLC, nes tokie valdikliai turi ribotus skaičiavimo išteklius. Tačiau pastaruoju metu atsirado įrankių, leidžiančių vykdyti neuroninius skaičiavimus tiesiogiai PLC aplinkoje. Pavyzdžiui, sukurta atviro kodo *CODESYS* ML biblioteka, kuri suteikia bazinius dirbtinių neuronų tinklo funkcionavimo komponentus tiesiai IEC 61131-3 programoje [17]. Naudodami tokias bibliotekas, inžinieriai gali suformuoti neuroninį tinklą (nustatyti sluoksnių skaičių, neuronus, aktyvacijos funkcijas) ir jį apmokyti arba įkrauti jau iš anksto apmokyto tinklo svorius į PLC. Pavyzdžiui, biblioteka numato funkcijas svorių failo užkėlimui į valdiklį ir netgi dinaminį mokymą – teoriniame lygmenyje įmanoma, kad pats valdiklis veikimo metu koreguotų modelio koeficientus [17]. Vis dėlto praktiškai PLC našumas dažnai riboja didesnių tinklų treniravimą – tam reikėtų labai ilgo laiko arba galingesnio procesoriaus. Dėl to pramonėje dažniau modelio skaičiavimas paprastinamas arba perkeliamas už valdiklio ribų. Pavyzdžiui, jeigu galios modelis realizuotas kaip paprastas MLP su keliolika įėjimų ir vienu ar keliais paslėptais sluoksniais, jį galima efektyviai vykdyti ir PLC cikle, to visiškai pakanka kelių matricos daugybų atlikimui. Tuo tarpu jei modelis

reikalauja didelių išteklių, pavyzdžiui, sudėtingas CNN arba kelių šimtų neuronų LSTM, racionaliau jį paleisti išoriniame įrenginyje.

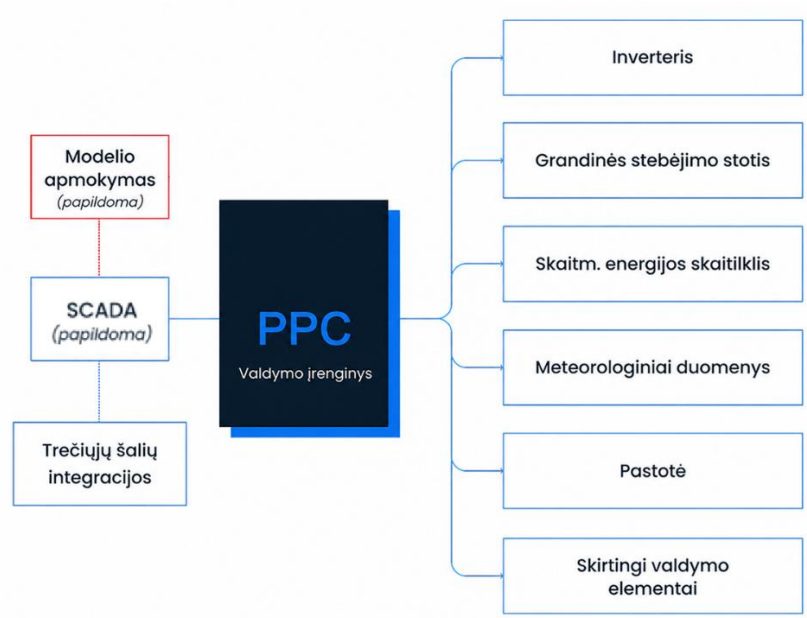
1.3.2. Decentralizuoti kraštinės kompiuterijos sprendimai: „Docker“, protokolai OPC UA, MQTT

Moderni automatika vis labiau naudojasi *edge* kompiuterijos principais – tai reiškia, kad duomenų apdorojimas atkeliamas arčiau lauko įrenginių, pasitelkiant vietinius pramoninius kompiuterius ar net pačius valdiklius. PV elektrinės atveju galima diegti atskirą kompiuterį, kuris būtų sujungtas su PLC valdikliu ir vykdytų skaičiavimams imlius modelio algoritmus realiu laiku. Tokia architektūra turi kelis privalumus:

1) Nuimama apkrovą nuo PLC, tuomet jis gali koncentruotis į kritinius valdymo uždavinius

2) Atsiranda galimybė naudoti pažangius programinius įrankius, konteinerius „Docker“ ir populiarias ML bibliotekas „TensorFlow“, „PyTorch“, nors šios paprastai nėra tiesiogiai suderinamos su PLC

3) Atsiranda lankstumas atnaujinant modelį – pakeisti ar perskaičiuoti algoritmą kraštinės kompiuterijos įrenginyje paprasčiau, nes tam nereikia stabdyti pagrindinio valdiklio darbo. Tokiam paskirstyta sprendimui įgyvendinti būtinas patikimas ryšys tarp modelio kompiuterio ir PLC.



2 pav. Paskirstyto modelio mokymo ir taikymo elektrinės valdymo sistemoje schema

Pramonėje įsitvirtino atvirieji protokolai, tokie kaip OPC UA ir lengvasis MQTT. OPC UA užtikrina standartizuotą duomenų apsikeitimą su struktūrizuota informacijos schema ir saugumo funkcijomis – jis plačiai naudojamas jėgainių valdymo sistemose, taip pat ir integruojant orų stotelių duomenis, SCADA ir kt. Pavyzdžiui viename iš naujausių atsinaujinančių šaltinių valdymo sprendimų aprašymų pažymima, kad integruotą DI modelių sistemą tikslinga realizuoti naudojant MQTT arba OPC UA komunikacijas – jos suteikia lengvai plečiamą ir patikimą duomenų perdavimą tarp posistemių, neapkraunant tinklo ir išvengiant duomenų praradimo [18]. MQTT dažnai naudojamas, kai reikia greito, lengvo ryšio tarp daugelio komponentų: pavyzdžiui, *edge* sistemoje gali veikti MQTT brokeris, per kurį DI modelis publikuoja apskaičiuotą PV galios reikšmę nustatytu dažniu, o PLC valdiklis yra MQTT klientas, prenumeravęs šiuos duomenis. Alternatyviai, PLC gali tiesiogiai

užklausyti modelio kompiuterį OPC UA protokolu kaip serverį ir nuskaityti reikiamas reikšmes. Šių protokolų privalumas – platforminis nepriklausomumas ir suderinamumas: daugelis šiuolaikinių valdiklių (pvz., „Siemens“, „Beckhoff“, „WAGO“ ir kt.) palaiko OPC UA, taip pat yra gausu MQTT klientų realizacijų. Todėl integruoti dockerizuotą modelį su tradiciniu PLC nėra sudėtinga.

Realiame projekte tai galėtų atrodyti taip: PV elektrinėje įrengta orų stotelė lokaliai teikia duomenis. Šie duomenys tiesiogiai pasiekiami PLC ar *edge* kompiuteriui. *Edge* kompiuteryje veikia DI modelio serveris, kuris nuolat pasiima atnaujintus duomenis per OPC UA ar Modbus tiesiai iš PLC ar orų stotelės [19], įveda juos į neuroninį tinklą ir suskaičiuoja PV galios maksimumą. Tada rezultatas perduodamas atgal PLC – pavyzdžiui, per tą patį OPC UA, kur PLC tiesiog nuskaito kintamąjį. PLC naudodamas šią reikšmę palygina ją su esama momentine galia ir, jei reikia, koreguoja keitiklių gamybos lygį, kad rezervas būtų išlaikytas, tarkim generacija laikoma ties 0,9 modeliuojamos maksimalios galios. Duomenų bazė (pvz., „InfluxDB“) gali kaupti visą istoriją: faktinę generaciją, modelio spėjimus, matuotus parametrus. Tai leidžia periodiškai atlikti modelio patikslinimą, modelis persimoko su atnaujintais duomenimis. Naujai ištreniruoto modelio svoriai tuomet automatiškai pradedami naudoti tolesnei prognozei. Tokiu būdu sistema vykdo nuolatinį mokymąsi, gerindama tikslumą. Pavyzdžiui, pradžioje rankiniu derinimu sudarytas 1-os eilės modelis gali turėti, tarkime, 10 % paklaidą tam tikromis valandomis, tačiau per kelis mėnesius DI modelis ją gali sumažinti iki kelių procentų. Tai tiesiogiai verčiasi nauda: mažesnė paklaida reiškia, kad elektrinė gali drąsiau išnaudoti turimą rezervą (nebijo, jog deklaruos daugiau nei išgalės pagaminti) – vadinasi, mažiau prarandama potencialios gamybos. Tyrimai rodo, kad pažangi rezervo valdymo strategija, paremta tiksliais modeliais, leidžia sumažinti rezervui reikalingą ribojimą net ~40 % palyginus su įprastine strategija [2]. Kitaip tariant, išmanus modelis leidžia rezervą tiekti tik tiek, kiek reikia, ir tuo pat metu maksimaliai gaminti elektros, kai tik galima. Be to, tikslus galios modelis yra būtinas įgyvendinant tokius pažangius tinklo stabilizavimo funkcionalumus kaip [7]:

1) Sintetinė inercija - keitiklis trumpam padidina galią staigaus dažnio kritimo metu išnaudodamas žinomą galios rezervą

2) Automatinis dažnio atkūrimo rezervas aFRR elektros rinkos balansavimo mechanizmuose.

3) Rankinis dažnio atkūrimo rezervas mFRR reikalingas elektros rinkos balansavimo mechanizmuose.

4) POD funkcija – dažnio valdymo funkcija, kuri mažina aktyviosios galios P arba reaktyviosios galios Q svyravimus elektros tinkle, kai sistema ar tinklas pradeda svyruoti

Visos šios funkcijos reikalauja pasitikėti elektrinės galimybėmis, o pasitikėjimas atsiranda tik turint moksliškai pagrįstą modelį su realaus laiko duomenų užtikrinimu.

1.3.3. Testavimas ir diegimo aspektai

Prieš pritaikant modelį pilnos apimties elektrinėje, svarbu jį kruopščiai patikrinti. Dažnai kuriamas skaitmeninis dvynys – virtualus elektrinės modelis, kuris imituoja panelių, keitiklių dinamiką. Į šį modelį integruojamas naujasis realaus laiko galios vertinimo algoritmas ir tikrinama, kaip jis veikia įvairiose scenarijuose (pvz., staiga užėjus debesims, esant neįprastai modulių temperatūrai ir pan.). Skaitmeninis dvynys leidžia saugiai įvertinti modelio robustiškumą ir, be kita ko, patikrinti, ar valdiklio reakcijos nekelia nepageidaujamų viršijimų ar svyravimų. Tik po tokių arba programinių

testų modelis diegiamas į realią PLC sistemą. Diegimo metu atsižvelgtina į kibernetinio saugumo klausimus. Galiausiai, eksploatuojant modelį, rekomenduojama turėti sutrikimų aptikimo mechanizmus: pvz., jei modelio apskaičiuota maksimali galia žymiai skiriasi nuo faktiškai pasiektos, sistema galėtų sugeneruoti aliarmą – neteisingo eksploatavimo prevencijai. Tai gali indikuoti tiek paties modelio netikslumą, tiek fizinę problemą su elektrine, pavyzdžiui sugedo dalis modulių, ir modelis pervertina galią. Tokiu būdu modelis tampa ne tik valdymo įrankiu, bet ir diagnostikos dalimi. Sujungus modeliavimą ir faktinius duomenis skaitmeninio dvynio platformoje, galima pasiekti iki ~95 % tikslumo gedimų prognozavime [20].

1.4. Literatūrinės apžvalgos apibendrinimas

Apibendrinant, galima teigti, kad fotovoltinės elektrinės maksimalios galios realaus laiko modeliavimo sprendimai yra kritiškai svarbūs modernioje elektros energetikos sistemoje, kurioje vis didesnį vaidmenį vaidina kintantys atsinaujinantys šaltiniai. Istoriškai pradėję nuo paprastų fizinių skaičiavimų, šie modeliai evoliucionavo iki pažangių hibridinių sistemų, kurios išnaudoja tiek fundamentalius PV elementų dėsnius, tiek ir mašininio mokymosi galią. Naujausi moksliniai tyrimai rodo, kad gerai suderinti DI modeliai gali užtikrinti labai aukštą tikslumą (viršijantį tradicinių metodų tikslumą) ir tuo pačiu būti pakankamai greiti realaus laiko valdymui [15]. Kartu pabrėžiama, jog optimaliausi rezultatai pasiekiami derinant metodus ir nuolat adaptuojant modelius prie naujų duomenų [14]. Praktikoje modelio integracija įgyvendinama arba tiesiogiai PLC valdikliuose, mažiems modeliams, arba naudojant paskirstytosios kompiuterijos principus – pvz., *edge* įrenginiuose su „Docker“ konteineriais, kas užtikrina suderinamumą ir patikimumą pramoninėje aplinkoje [18]. Toks skaitmeninės ir valdymo technologijų derinys leidžia PV elektrinėms ne tik tiekti žalią energiją, bet ir aktyviai dalyvauti tinklo valdyme: teikti dažnio atkūrimo rezervus, imituoti inerciją, stabilizuoti įtampą. Galų gale, tai didina atsinaujinančių šaltinių integracijos lubas ir ekonomiškai naudingai išplečia jų panaudojimą. Galima tvirtinti, kad realaus laiko PV galios modeliavimas yra intensyviai tyrinėjama sritis, kurioje susilieja energetikos inžinerija, automatikos sprendimai ir dirbtinis intelektas – būtent toks tarpdisciplininis požiūris būtinas kuriant naujos kartos išmaniuosius elektros tinklus.

2. Metodinė darbo dalis saulės elektrinės generuojamos galios prognozavimo modelių kūrimui ir integracijai į PLC valdymo sistemą

2.1. Tyrimo logika ir darbo tikslas

Magistro darbo metodinė dalis bus grindžiama ne abstrakčiu laboratoriniu prognozavimo uždaviniu, o realios pramoninės fotovoltinės elektrinės valdymo poreikiu: realiu laiku įvertinti maksimaliai prieinamą generuojamą galią taip, kad šis įvertis būtų tinkamas PPC logikai, galios rezervo formavimui, tinklo paslaugoms ir atitikties bandymams. Tyrimo objektas yra didelės galios elektrinė, kurioje aktualus ne vien statistinis tikslumas, bet ir modelio praktinis tinkamumas eksploatacijai. Baigiamojo darbo uždaviniai numato modelių apžvalgą, kūrimą, konteinerizuotą infrastruktūrą, integraciją į pramoninį valdiklį ir eksperimentinį palyginimą su dabar taikomais sprendimais.

Toks metodinis pasirinkimas atitinka naujausią literatūrą: naujausios apžvalgos pabrėžia, kad PV galios prognozavimas tampa kritiškai svarbus tinklo planavimui, dispečeriniam valdymui, rezervų poreikiui ir patikimam saulės generacijos integravimui, o modelių vertinimas turi apimti ne tik architektūrą, bet ir pritaikomumą konkrečiam laiko horizontui, oro sąlygoms, duomenų kokybei ir realaus laiko naudojimui [21].

Todėl šiame darbe eksperimentas bus planuojamas dviem sluoksniais. Pirmasis sluoksnis bus plačioji modelių generavimo ir palyginimo fazė, kurioje bus tikrinami kelių klasių modeliai: fizikinis bazinis metodas, klasikiniai statistiniai modeliai, klasikiniai mašininio mokymosi modeliai, neuroniniai tinklai ir, jei reikės, hibridiniai modeliai. Antrasis sluoksnis bus siauroji diegimo fazė, kurioje į PLC bus perkeliami tik tie modeliai, kurie kartu tenkins tikslumo, greitaveikos, struktūrinio paprastumo ir eksploatacinio patikimumo reikalavimus. Būtent toks atskyrimas tarp „tiksliausio laboratorijoje“ ir „geriausio realiai eksploatacijai“ yra rekomenduojamas ir naujausiose PV prognozavimo apžvalgose [21].

Praktinis darbo tikslas yra sumažinti prognozės paklaidą lyginant su šiuo metu taikomu Laplaso perdavimo funkcija paremtu metodu, kartu išlaikant modelį pakankamai lengvą, kad jis galėtų būti patikimai vykdomas PLC cikle ir periodiškai atnaujinamas iš debesijos ar kito centralizuoto Linux serverio [22].

Praktikoje taikytas PV elektrinės maksimalios galios nustatymo metodas (įmonės kontekste) buvo paremtas meteorologiniais jutikliais:
- Piranometro matuojama apšvieta (W/m²);
- Saulės modulių temperatūra (C°).

Šie duomenys naudojami kaip pagrindinės įvestys į pirmojo laipsnio Laplaso perdavimo funkciją, pagal kurią realiu laiku buvo skaičiuojamas teorinis maksimalios generacijos potencialas.

$$P_{available}(t) = \left[\frac{K_{Irr}}{T_{Irr}s+1} * Irr(t)\right] * \left[1 + K_{Temp}(Temp_{ref} - Temp(t)\right]; \qquad (1)$$

Čia: $P_{available}(t)$ – modeliuojama galia (kw), Irr(t) – momentinė saulės spinduliuotė (W/m²), Temp(t) – momentinė PV modulio temperatūra (C°), $Temp_{ref}$ – optimalaus PV modelio darbo taško temperatūra (C°), K – stiprinimo koeficientas, T – laiko pastovioji (s)

Teoriškai idealiomis sąlygomis toks sprendimas - priimtinas: PV galios priklausomybė nuo apšvietos ir temperatūros aprašoma paprastesnėmis funkcijomis, o dinamika modeliuojama perdavimo funkcija Laplaso srityje. Šios struktūros privalumas – aiški ir lengvai realizuojama dinamika, leidžianti

kontroliuoti modelio greitaveiką, tačiau praktikoje modelio parametrai turi būti identifikuojami ir periodiškai perkalibruojami pagal realius duomenis. Taikant minėtą metodą, praktinis kalibravimas buvo atliekamas rankiniu būdu parenkant K ir T, o kelių piranometrų ir temperatūros daviklių duomenys buvo surenkami, filtruojami ir išvedamas jų vidurkis.

1 lentelė. Planuojami lyginti modeliai

Modelių grupė	**Paskirtis eksperimente**	**Pagrindinė vertinimo logika**	**Numatomas vaidmuo diegime**
Laplaso perdavimo funkcija	Bazinis atskaitos taškas	Palyginimas su dabar taikomu fizikinės logikos sprendimu	Visada naudojamas kaip bazinė kreivė
MLP	Pagrindinis kompaktiškas DI kandidatas	Tikslumas, paprastumas, svorių eksportavimo galimybė, PLC vykdomumas	Pirminis kandidatas galutiniam diegimui
RF	Klasikinio ML etalonas	Ar nelinijinis medžių modelis lenkia MLP tikslumu	Diegiamas tik jei modelio dydis ir perdavimas lieka praktiški
ARIMA	Statistinis etalonas	Ar pakanka autoregresinės logikos, kai turimas grįžtamasis ryšys	Naudojamas kaip bazinis laiko eilučių palyginimas
LSTM ir hibridiniai modeliai	Tyriminė „viršutinė riba“	Ar tikslumo prieaugis pateisina didesnę kainą ir sudėtingumą	Dažniausiai lieka tiriamajam palyginimui

2.2. Eksperimentinė aplinka ir infrastruktūra

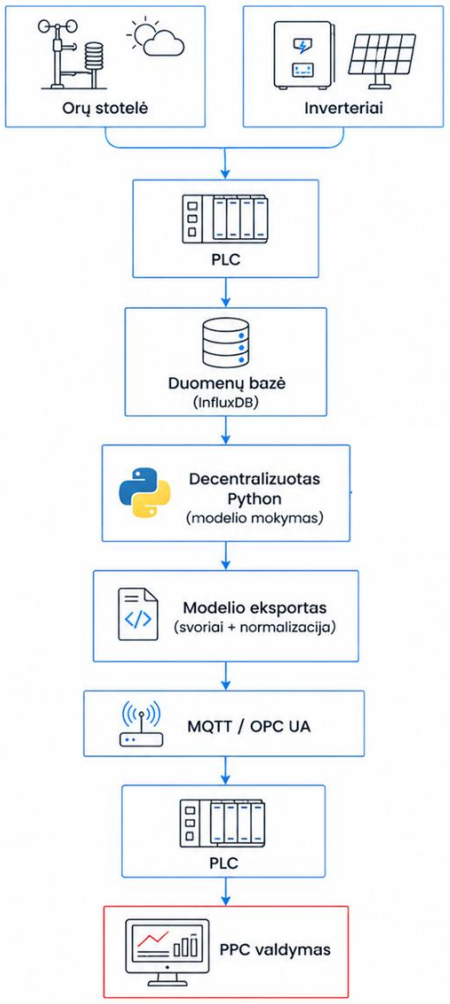
Modelių kūrimo aplinka bus organizuojama Python pagrindu, projektas kuriamas „PyCharm“ aplinkoje, versijavimui naudojamas „GitHub“, duomenų apdorojimui naudojamos „NumPy“ ir „Pandas“ bibliotekos. Metodiškai tai yra tinkamas pasirinkimas, nes tokia grandinė leidžia atkartojamai valdyti duomenų paruošimą, modelių iteracijas ir eksportuojamų parametrų generavimą.

Vykdymo dalis bus planuojama naudojant „WAGO Edge Controller“ valdiklį. „WAGO“ dokumentacijoje nurodoma, kad šios klasės įrenginiai yra programuojami per CODESYS programinę aplinką, turi iš anksto įdiegtą Linux pagrindu veikiančią aplinką ir yra sukurti integracijai į atviras IoT architektūras. PLC pusėje vykdomas lengvas apskaičiavimo sluoksnis, kas yra be techniškai didelių resursų [23].

Duomenų ir modelio parametrų perdavimui bus naudojama MQTT sąsaja. CODESYS IoT bibliotekos dokumentacijoje nurodoma, kad „MQTT Client SL“ leidžia valdikliui komunikuoti su MQTT brokeriu, naudoti prenumeravimo / publikavimo *(angl. subscribe / publish)* logiką ir perduoti JSON eilutes. Papildomai 2024 m. publikuotas pramoninis OPC-UA ir MQTT palyginimo tyrimas parodė, kad kai yra reikalingas greitas ir lengvas duomenų perdavimas be sudėtingo aprašymo, MQTT kaip lengvesnis protokolas turi daug greitaveikos, paprastumo ir kt. pranašumų. Tai tiesiogiai pagrindžia sprendimą modelio parametrus ir susijusius metaduomenis perduoti į PLC per MQTT [24].

Todėl bendra eksperimentinė infrastruktūra bus organizuojama taip:

– Istoriniai duomenys bus paimami iš laiko eilučių duomenų bazės;
– Modelių mokymas ir permokymas vyks centralizuotai Python aplinkoje;
– Eksportuojami normalizavimo parametrai ir modelio svoriai bus suformuojami į bendrą perdavimo struktūrą, išsiunčiami MQTT ryšiu;
– PLC pusėje dekoduojami į *„CODESYS“* vykdomą logiką.



3 pav. Preliminari sistemos architektūros schema

Ši schema leis palyginti ne tik modelių tikslumą, bet ir atnaujinimo procesą, žinučių patikimumą, bei vietinio vykdymo stabilumą

2.3. Duomenų paruošimo ir požymių formavimo metodika

Tyrime bus naudojami realios pramoninės saulės elektrinės telemetrijos duomenys. Pasirinktas objektas yra viena didžiausių PV elektrinių Lietuvoje, kurios instaliuota galia siekia apie 58 MW, leidžiama generuoti galia — apie 50 MW, o elektrinė papildomai integruota su apie 74 MW galios ir apie 120 MWh talpos BESS sistema. Istorinių duomenų kaupimui naudojama „InfluxDB“, o duomenys sistemingai kaupiami nuo 2024 m. gruodžio, leidžiant dirbti tiek su aukšto dažnio 1 Hz matavimais, tiek su ilgesnio laikotarpio sezoninėmis tendencijomis. Toks duomenų šaltinis metodologiškai yra labai vertingas, nes leidžia viename tyrime sujungti modelių mokymą ir validaciją realiomis eksploatacijos sąlygomis.

Tikslinis išėjimo kintamasis bus aktyvioji PV generuojama galia, o įėjimo kintamųjų struktūra bus formuojama iš meteorologinių ir laiko požymių, sinchronizuotų bendroje laiko ašyje. Mokymo duomenys bus imami SQL užklausomis iš laiko eilučių bazės, suvienodinami iki vienodo dažnio, po to filtruojami taip, kad liktų fiziškai prasmingos būsenos. Tai apims neigiamų, akivaizdžiai klaidingų ir reikšmių, kada elektrinė yra ribojama duomenų šalinimą, trūkstamų taškų tvarkymą ir matavimų sujungimą pagal laiko žymą. Toks duomenų rengimas yra būtinas, nes PV prognozės paklaida dažnai kyla ne tik iš pasirinkto modelio, bet ir iš netiksliai suvestų, nesuvaldytų ar fiziškai neadekvačių signalų [21].

Požymių formavimo etape bus taikomas ciklinių laiko kintamųjų kodavimas sinuso ir kosinuso poromis, nes tai leidžia modeliui taisyklingai interpretuoti paros ir metų ciklius kaip tęstinius, o ne laužytus intervalus. Modelių mokymui bus parenkami tokie požymiai kurie yra ir informatyvūs ir prasmingi PLC diegimo požiūriu. Tai reiškia, kad prioritetas bus teikiamas dinamiškiems požymiams — apšvietai, temperatūrai bei cikliniams laiko kodams — o statiški ar mažai kintantys požymiai bus paliekami tik tada, jei duos aiškų paklaidos sumažėjimą. Toks sprendimas dera su 2025 m. tyrimu, kuriame parodyta, kad sistemingas požymių atrinkimas gali vienu metu sumažinti modelio sudėtingumą ir pagerinti prognozavimo tikslumą, taip pat su 2025–2026 m. apžvalgomis, kuriose meteorologiniai bei istoriniai PV duomenys įvardijami kaip dažniausi ir svarbiausi įėjimai [25]. Dėl to magistrinio darbo eksperimente duomenų kokybės tikrinimas bus laikomas atskira metodikos dalimi, o galutinė kompaktiška įėjimų struktūra bus tvirtinama tik įsitikinus, kad visi pasirenkami signalai yra laike stabilūs, sinchroniški ir fiziškai interpretuojami. Testavimas bus atliekamas atsižvelgiant į sezonines validavimo atkarpas. Naujausios apžvalgos pabrėžia, kad PV modelių našumas reikšmingai priklauso nuo oro kaitos, klimato ir sezoniškumo, o 2025 m. tyrime apie skaitmeninius dvynius parodyta, jog net ir gerai veikiančio MLP ar fizikinio modelio pranašumas gali kisti priklausomai nuo sezono. Todėl rezultatų tikrinimas bus atliekamas bent jau atskirai žiemos ir vasaros laikotarpiams, bei atskirai dienos režimams, kuriuose vyksta staigūs apšvietos pokyčiai [26].

2.4. Modelių generavimo ir atrankos metodika

Modelių kūrimo fazėje bus laikomasi principo, kad vienas modelis nėra iš anksto priimamas kaip „teisingas“ — jis turi laimėti atranką pagal bendrą kriterijų rinkinį. Pagal baigiamojo darbo uždavinius bus lyginami skirtingi prognozavimo principai: esamas fizikinis-Laplaso modelis, autoregresinis statistinis modelis, klasikiniai mašininio mokymosi modeliai, MLP tipo neuroninis tinklas, sekų modeliai ir, esant poreikiui, hibridiniai modeliai. Naujausios 2024–2026 m. apžvalgos būtent taip ir siūlo struktūruoti PV prognozavimo tyrimus — pagal fizinius, statistinius, ML ir hibridinius metodus, nes tik tokia schema leidžia parodyti ne tik pavienio algoritmo, o visos metodų klasės vietą konkrečiame taikyme [21].

Apmokymo procedūra bus vykdoma iteratyviai. Kiekvienam modeliui bus formuojamas standartizuotas duomenų paruošimo vamzdynas *(angl. „pipeline“),* apibrėžiamos parametrų paieškos ribos, o visi eksperimentų rezultatai bus versijuojami, užtikrinant pilną atsekamumą tarp iteracijų. Toks metodas grindžiamas ne vienkartiniu optimalaus sprendimo radimu, bet kartotine optimizavimo ir modelio tobulinimo paieška. Literatūra pabrėžia, kad PV prognozavimo tikslumui didelę įtaką daro pakartotinė požymių atranka ir optimizavimo strategija. Praktine prasme tai reiškia, kad modelio architektūros, mokymo lango, normalizavimo ir požymių rinkinio pakeitimai bus vertinami kaip atskiri eksperimentiniai veiksniai, o ne kaip atsitiktiniai kodo pakeitimai [26].

Darbo metu pagrindinis dėmesys bus skiriamas MLP klasei. Tokį pasirinkimą pagrindžia ir naujausi tyrimai: 2025 m. darbe apie PV skaitmeninius dvynius MLP pasiekė geriausią bendrą tikslumą ir sezoninį stabilumą, o požymių atrankos tyrime parodyta, kad MLP, derinamas su tinkama požymių atranka, gali pasiekti labai stiprius nMAE ir R^2 rodiklius neperaugdamas į per daug sudėtingą architektūrą. Todėl pagrindinis darbo kandidatas bus kompaktiškas MLP, kurio architektūrų šeima bus tikrinama taip, kad būtų rastas geriausias balansas tarp tikslumo, svorių skaičiaus ir PLC realizavimo paprastumo [26].

LSTM ir kiti gilesni modeliai nebus atmetami iš anksto, tačiau jų vaidmuo šiame darbe bus pirmiausia eksperimentinis. Tyrimai rodo, kad LSTM šeimos modeliai literatūroje naudojami labai dažnai, tačiau kartu akcentuojamas poreikis kurti lengvesnius, interpretuojamus ir realiai diegiamus sprendimus. 2026 m. hibridinių AI modelių apžvalga tiesiogiai nurodo, kad nors hibridai dažnai būna tikslūs, jų diegimą riboja duomenų kokybė, skaičiavimo kaina ir interpretacijos stoka, todėl šiame darbe svarbu vertinti ne vien žemiausią paklaidą, bet ir pasirengimą realiam naudojimui [27].

RF ir ARIMA šiame darbe bus laikomi svarbiais modeliais. RF leis patikrinti, ar klasikinis nelinijinis ML modelis gali suteikti tikslumo pranašumą prieš kompaktišką MLP, o ARIMA bus naudinga parodyti, kiek prognozavimo kokybė priklauso nuo autoregresinės grįžtamojo ryšio logikos. Naujausiuose darbuose matyti, kad klasikiniai ir pažangesni ML metodai tebėra konkurencingi PV galios uždaviniuose, tačiau kartu literatūroje akcentuojama, jog realiame taikyme svarbi ir mokymo kaina, ir atnaujinimo sudėtingumas. Todėl šie modeliai darbe bus naudojami ne tam, kad „būtinai laimėtų“, o kad parodytų metodinį kontrastą tarp skirtingų neuroninių tinklų [28].

Galiausiai, jei eksperimentinėje fazėje paaiškėtų, kad tam tikri hibridiniai sprendiniai duoda reikšmingą pranašumą į PLC realizavimo fazę pateks tik tada, jei jų struktūra bus pakankamai lengva. Tai dera su hibridinių modelių ir *edge* prognozavimo tyrimais, kurie rodo, kad optimizuoti hibridai metodai dažnai duoda gerą tikslumo ir efektyvumo santykį, bet realaus naudojimo scenarijuose reikia ypač vertinti resursų kainą ir diegimo paprastumą [29].

2.5. PLC diegimo ir realaus laiko validavimo metodika

Tyrimo metu modelių palyginimas nebus laikomas pakankamu galutiniam sprendimui priimti. Galutinis metodikos etapas bus modelio perkėlimas į PLC ir paralelinis testavimas realioje elektrinėje. Darbo užduotis aiškiai rodo, kad tokia logika laikoma esmine – modelis turi būti lyginamas ne tik su realia aktyviąja galia, bet ir su šiuo metu taikomu Laplaso metodu, o svarbiausi kriterijai yra ne vien paklaida, bet ir stabilumas PLC cikle bei infrastruktūrinis tinkamumas eksploatacijai.

Šiame darbe sprendimas bus projektuojamas pagal esamos pramoninės valdymo sistemos apribojimus. PLC nėra galingas serveris ir jo pagrindinė funkcija yra deterministinis valdymas, o ne didelių neuroninių tinklų vykdymas. Dėl to sudėtingesni modeliai bus testuoti kaip teorinis tikslumo potencialas, kad būtų galima įvertinti, kiek daugiau tikslumo būtų galima pasiekti ateityje, jei būtų naudojamas galingesnis *edge* įrenginys ar kitas galingesnės programinės įrangos sprendimas.

Metodiškai diegimas bus organizuojamas per bendrą eksportavimo sluoksnį. Tai reiškia, kad modelių, patekusių į PLC validavimo etapą, parametrai bus transformuojami į bendrą perdavimo struktūrą:
- modelio tipą;
- normalizavimo konstantas;

– architektūros informaciją;
– parametrų masyvą.

Tokia schema leis į PLC siųsti ne tik padraiką informaciją, o standartizuotą parametrų rinkinį, kurį CODESYS pusėje galės iškoduoti parašytas funkcinis blokas. Tai leis sąžiningai palyginti kelis modelius toje pačioje programoje, nes skirsis tik regresoriaus matematika, o ne aplinkinė infrastruktūra. Šį pasirinkimą techniškai palaiko *CODESYS* ir MQTT galimybės [24].

Realaus laiko bandymo metu duomenų bazėje bus kaupiamos sinchronizuotos palyginimų kreivės:
– reali aktyvioji PV galia;
– Laplaso metodo išėjimas;
– PLC veikiančio modelio prognozė;

Esminė validavimo taisyklė bus ta, kad lyginimas turi vykti sinchronizuotais laiko taškais ir toje pačioje eksploatacijos aplinkoje. Tokiu būdu bus galima atskirai identifikuoti atvejus, kada modelis tiksliai seka generacijos dinamiką, ir atvejus, kada atsiranda vėlavimas, sisteminis pervertinimas, neigiami ar kitaip nefiziški išėjimai.

Papildomai bus atliekamas vykdymo validavimas PLC pusėje. Jis apims dekodavimo korektiškumo patikrą, ciklo stabilumą, prognozės ribojimą iki fiziškai prasmingų reikšmių ir pranešimų perdavimo patikimumo įvertinimo. Tokie kriterijai yra svarbūs todėl, kad net ir tikslus modelis gali būti netinkamas, jei jį sunku atnaujinti, jei jo perdavimas priklauso nuo trapių rankinių veiksmų. Literatūroje būtent toks kompromisas tarp tikslumo ir vykdymo kainos yra laikomas vienu svarbiausių kriterijų [30].

Šis metodinis pasirinkimas dera ir su Lietuvos eksploatavimo bei atitikties kontekstu. „ESO“ viešai nurodo, kad natūriniai atitikties bandymai ir su jais susijusi dokumentacija yra aktualūs elektrinių eksploatacijai, o elektrinių ir energijos kaupimo įrenginių, kurių įrengtoji galia didesnė nei 100 kW, valdymo sistemos turi atitikti saugumo reikalavimus. Todėl šiame darbe realaus objekto testas bus projektuojamas taip, kad sugeneruotų pakankamai aiškų, atsekamą ir pakartojamą duomenų rinkinį ne tik akademinei analizei, bet ir praktiniam sistemos pagrindimui [6].

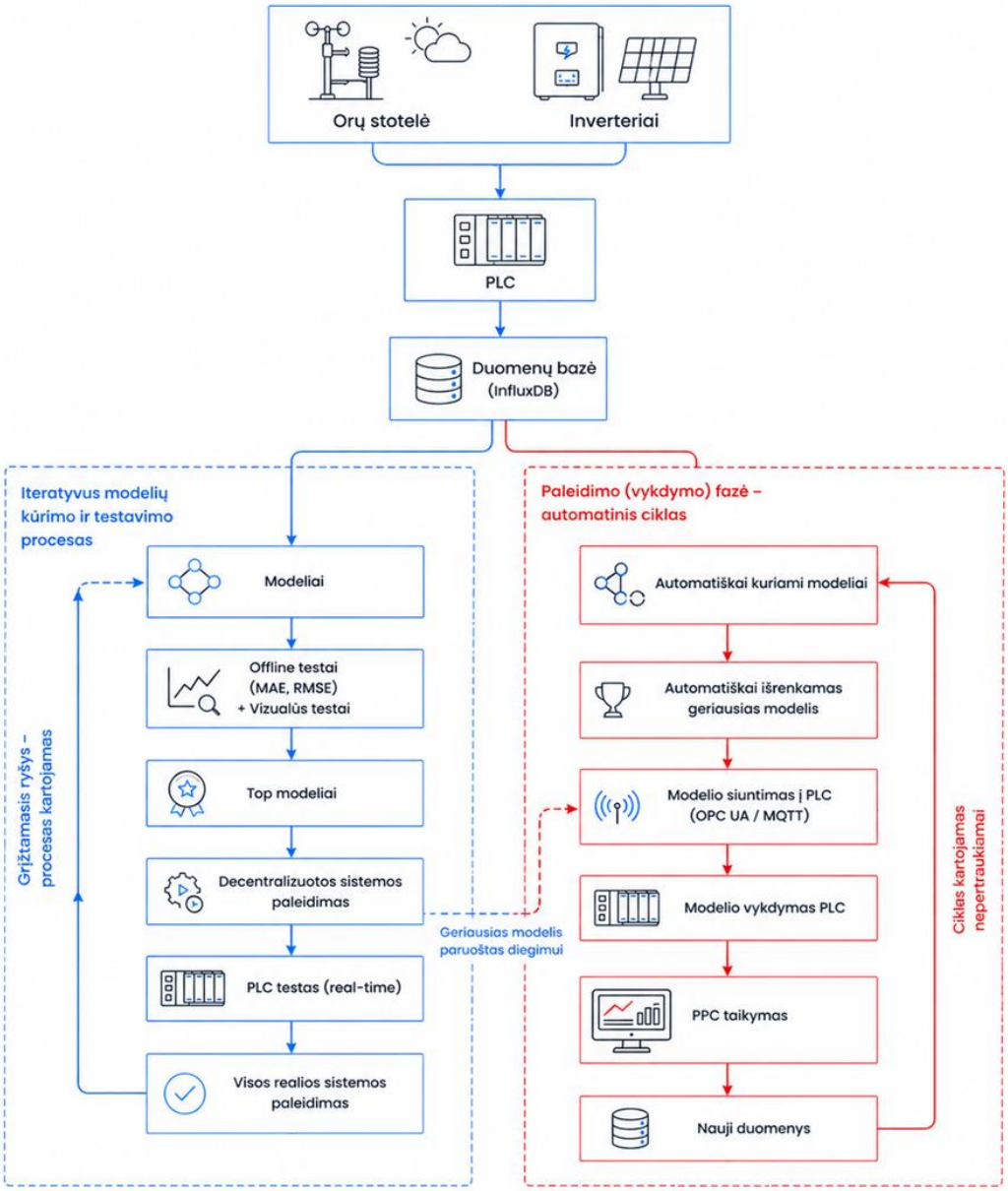
2.6. Rezultatų lyginimo ir interpretavimo metodika

Rezultatų palyginimas bus atliekamas keliuose sluoksniuose. Pirma, kiekvienas kandidatinis modelis bus vertinamas *„offline“* režimu pagal sinchronizuotas istorines imtis. Antra, į PLC perkeltas modelis bus vertinamas *„online“* režimu, lyginant jo prognozę su realia aktyviąja galia ir baziniu Laplaso metodu. Trečia, galutinis sprendimas bus priimamas ne pagal vieną skaičių, o pagal kelių kriterijų visumą: paklaidą, stabilumą, greitaveiką, modelio dydį, atnaujinimo paprastumą ir elgseną nestacionariuose režimuose. Būtent tokį daugiakriterį vertinimą rekomenduoja ir naujausios PV prognozavimo apžvalgos [21].

Pagrindiniu vertinimo rodikliu bus laikomas MAE, nes jis aiškiai interpretuojamas vienetais ir tiesiogiai parodo vidutinį absoliutų nuokrypį nuo realios aktyviosios galios. Papildomai bus skaičiuojami RMSE ir MSE, nes šie rodikliai jautriau vertina dideles lokalias klaidas, kurios ypač svarbios staigių apšvietos šuolių metu. Toks rodiklių rinkinys sutampa su plačiai naudojama saulės prognozių praktika [31]. Tačiau vien kiekybiniai rodikliai nebus laikomi pakankamais. Bus atliekama ir vizualinė kreivių analizė, nes pramoniniam naudojimui svarbu ne tik vidutiniškai maža paklaida,

bet ir tai, ar modelis teisingai atkuria kreivės formą, ar neatsilieka nuo realių šuolių, ar nekuria nefiziškų elgsenų saulei leidžiantis ir ar nepervertina prieinamos galios debesuotumo metu. Todėl magistrinio darbo metodinėje dalyje jis bus formaliai įtrauktas kaip privalomas vertinimas.

Vertinimas bus atliekamas atskirai keliems režimams: žiemos ir vasaros periodams, tipiškoms saulėtoms dienoms, kintamo debesuotumo dienoms, rytinio kilimo ir vakarinio kritimo atkarpoms bei paros dalims, kai reali generacija artėja prie nulio. Toks skaidymas būtinas todėl, kad literatūra nuosekliai rodo tiek meteorologinių sąlygų, tiek sezono įtaką modelio veikimui, o 2025–2026 m. apžvalgos aiškiai pabrėžia, jog modelio pasirinkimas negali būti atskirtas nuo prognozavimo scenarijaus ir vykdymo konteksto [21].



4 pav. Preliminari darbo eigos schema

Galutinis modelis šiame darbe bus laikomas geriausiu tik tada, jei vienu metu tenkins tris sąlygas. Pirma, jis turės mažesnę paklaidą už bazinį Laplaso metodą pagrindiniuose darbo scenarijuose. Antra, jo išėjimas bus fiziškai prasmingas ir stabilus realiame PLC vykdyme. Trečia, jo integravimas,

atnaujinimas ir palaikymas bus paprastesnis. Kitaip tariant, šiame darbe bus renkamasi ne matematiškai tiksliausias, o eksploatacijai geriausias modelis [29].

2.7. Metodinės darbo dalies apibendrinimas

Apibendrinant, metodinėje darbo dalyje buvo suformuota tyrimo eiga, leidžianti saulės elektrinės generuojamos galios prognozavimo uždavinį vertinti ne tik kaip matematinio modelio kūrimą, bet ir kaip realiai diegiamą pramoninės valdymo sistemos sprendimą. Dėl to modelių kūrimo metodika buvo paremta dviem pagrindiniais principais:

– pirmiausia modeliai turi būti objektyviai palyginti pagal prognozavimo tikslumą;
– vėliau atrenkami tik tie sprendimai, kuriuos galima patikimai realizuoti PLC valdiklyje.

Metodinėje dalyje numatyta naudoti realios fotovoltinės elektrinės istorinius duomenis, apimančius aktyviąją galią, apšvietą, temperatūrą ir laiko požymius. Kadangi realūs telemetrijos duomenys gali būti nevienodai registruojami, turėti trūkstamų reikšmių ar apimti ribotos generacijos režimus, duomenų paruošimas laikomas viena svarbiausių tyrimo dalių. Numatytas signalų sinchronizavimas, filtravimas, ribojimo būsenų atmetimas ir ciklinių laiko požymių formavimas.

Modelių atrankai numatyta lyginti kelias skirtingas prognozavimo metodų grupes, t.y. esamą Laplaso transformacijos pagrindu veikiantį metodą, MLP neuroninį tinklą, RF, ARIMA, LSTM ir kitus sekų arba hibridinius modelius. Toks palyginimas leidžia įvertinti ne tik atskirų modelių tikslumą, bet ir jų tinkamumą pramoniniam diegimui. Galutinis modelis šiame darbe turi būti vertinamas ne vien pagal mažiausią MAE ar RMSE reikšmę, bet ir pagal skaičiavimo paprastumą, parametrų kiekį, atnaujinimo galimybę ir stabilumą realaus laiko PLC cikle.

Taip pat metodinėje dalyje pagrįsta decentralizuota modelio kūrimo architektūra. Modelių mokymas numatytas *Python* aplinkoje arba Linux VM serveryje, o PLC valdiklyje paliekamas tik apmokyto modelio vykdymas. Modelio parametrų perdavimui pasirinkta MQTT sąsaja, leidžianti automatizuotai perduoti modelio tipą, normalizavimo koeficientus ir parametrų masyvą į valdiklį. Tokia struktūra sudaro sąlygas periodiniam modelio atnaujinimui nekeičiant PLC programos logikos.

Taigi metodinė darbo dalis apibrėžia nuoseklų kelią nuo duomenų paruošimo ir modelių mokymo iki jų integracijos į realią elektrinės valdymo sistemą. Ši metodika leidžia spręsti ne tik prognozavimo tikslumo, bet ir praktinio pritaikomumo klausimą, todėl galutinis sprendimas turi būti suprantamas kaip tikslumo, paprastumo ir eksploatacinio patikimumo kompromisas.

3. Tiriamoji dalis

Eksperimentinė darbo dalis buvo formuojama ne tik kaip saulės elektrinės generuojamos galios prognozavimo modelių tikslumo tyrimas, bet ir kaip praktiškai pritaikomos sistemos kūrimas. Kadangi darbo uždaviniuose numatyta ne tik sukurti ir palyginti skirtingus prognozavimo modelius, bet ir integruoti juos į pramoninį valdiklį, tyrimo eiga nuo pradžių buvo orientuota į realaus eksploatavimo sąlygas. Dėl to buvo siekiama kurti tokią programinę infrastruktūrą, kuri vėliau galėtų būti konteinerizuota, automatiškai apmokyti modelius, eksportuoti jų parametrus ir perduoti juos į PLC valdiklį.

Tyrimo objektu pasirinkta reali pramoninė fotovoltinė elektrinė Lietuvoje. Ankstesnėje darbo dalyje nurodyta, kad tiriamos elektrinės instaliuota galia siekia apie 50 MW, o leidžiama generuoti galia – apie 40 MW. Taip pat elektrinė susieta su BESS sistema, kurios instaliuota galia apie 74 MW, o talpa apie 120 MWh . Toks objektas pasirinktas todėl, kad reali eksploatacinė aplinka leidžia įvertinti modelio veikimą ne laboratorinėmis sąlygomis, o esant tikriems apšvietos, temperatūros, debesuotumo, technologinių ribojimų ir valdymo sistemos darbo režimų pokyčiams.

Pagrindinis tiriamosios dalies tikslas – sukurti ir eksperimentiškai įvertinti tokį generuojamos galios prognozavimo sprendimą, kuris būtų ne tik pakankamai tikslus, bet ir tinkamas naudoti realaus laiko valdymo sistemoje. Todėl modeliai vertinami ne tik pagal paklaidos metrikas, bet ir pagal jų praktinio integravimo galimybes: skaičiavimo paprastumą, parametrų perdavimą į PLC, vykdymo greitį, stabilumą ir galimybę modelį automatiškai atnaujinti.

3.1. Modelių mokymas naudojant „CODESYS ML“ biblioteką

Prieš pasirenkant galutinę modelių kūrimo ir diegimo kryptį buvo įvertinta galimybė modelius kurti arba apmokyti pačioje PLC aplinkoje. Šiam tikslui buvo analizuojama „CODESYS ML“ biblioteka, skirta mašininio mokymosi funkcionalumui realizuoti CODESYS aplinkoje. Kadangi tyrimo tikslas buvo sukurti realiai eksploatuojamą sprendimą, toks variantas pradiniame etape atrodė patrauklus: modelis galėtų būti kuriamas, apmokomas ir naudojamas tame pačiame valdiklyje, kuriame vykdoma elektrinės valdymo logika. Tokiu atveju būtų sumažintas priklausomumas nuo išorinių sistemų, o modelio išvestis galėtų būti naudojama tiesiogiai PLC cikle.

Vis dėlto detalesnė analizė parodė, kad modelio kūrimas pačiame PLC nėra tinkamiausias kelias šiam darbui. „CODESYS ML“ bibliotekos naudojimas reikalauja iš anksto parengtos struktūros, funkciniai blokai turi būti integruojami į valdiklio programą, o modelio mokymui ir taikymui reikalinga papildoma valdymo logika. Praktiniame tyrimo kontekste tai keltų kelias problemas. Pirmiausia, istoriniai elektrinės duomenys nėra kaupiami pačiame PLC valdiklyje – jie saugomi laiko eilučių duomenų bazėje „InfluxDB“ (įrašai kaupiami nuo 2024 m. gruodžio). Norint apmokyti modelį PLC aplinkoje, reikėtų papildomai realizuoti duomenų užklausimą iš duomenų bazės, jų paruošimą, filtravimą, normalizavimą ir pateikimą bibliotekai tinkamu formatu. Tai reikšmingai apsunkintų valdiklio programą.

Antra, PLC valdiklis yra skirtas realaus laiko valdymui, signalų apdorojimui, komunikacijai ir elektrinės valdymo funkcijų užtikrinimui. Modelių mokymas, ypač kai norima atlikti daug iteracijų, tikrinti skirtingas architektūras ir vertinti jų paklaidas, reikalauja didesnių skaičiavimo resursų ir lankstesnės programavimo aplinkos. Mokant modelius tiesiogiai PLC, būtų pernelyg apsunkintas. Be to, didesni neuroniniai tinklai ar sudėtingesni modeliai galėtų sukelti papildomą CPU apkrovą.

Trečia, „CODESYS ML“ bibliotekos taikymas ribotų modelių struktūrų pasirinkimą. Šiame darbe buvo svarbu ne tik sukurti vieną modelį, bet ir pereiti per daug iteracijų, lyginti skirtingus įėjimo požymius, architektūras ir modelių tipus. Dėl to buvo pasirinkta decentralizuota architektūra. Tokiu būdu PLC valdiklis naudojamas tik tam, kam jis tinkamiausias, o sunkesni modelio mokymo darbai perkeliami į atskirą infrastruktūrą.

3.2. Decentralizuotos modelių kūrimo architektūros pasirinkimas

Atsisakius modelio mokymo PLC aplinkoje, buvo pasirinkta architektūra, kurioje modelio kūrimas ir testavimas atliekamas Python aplinkoje, o PLC valdiklyje vykdomas tik paruošto modelio skaičiavimas. Toks metodas leidžia pasinaudoti Python ekosistemos privalumais, išlaikyti modelių testavimo lankstumą ir tuo pačiu neperkrauti PLC valdiklio pertekliniais mokymo ar duomenų apdorojimo procesais.

Pirmoji programos versija buvo kuriama „PyCharm“ programavimo aplinkoje. Ši aplinka pasirinkta dėl patogaus projektų valdymo, bibliotekų integracijos, kodo struktūrizavimo ir derinimo galimybių. Pirminė programos logika buvo orientuota į MLP tipo neuroninio tinklo sukūrimą naudojant „PyTorch“ biblioteką. Toks pasirinkimas buvo pagrįstas tuo, kad MLP yra santykinai paprasta, aiškiai interpretuojama ir PLC aplinkoje realizuojama neuroninio tinklo struktūra.

Pirminėje tyrimo stadijoje buvo svarbu ne iš karto pasiekti maksimalų prognozavimo tikslumą, o sukurti visą bazinę grandinę: duomenų nuskaitymą, jų paruošimą, MLP modelio apmokymą, modelio eksportavimą ir jo taikymo pakartojimą CODESYS aplinkoje. Tai leido patikrinti, ar Python aplinkoje apmokytas neuroninis tinklas gali būti perrašytas į PLC vykdomą logiką taip, kad CODESYS aplinkoje apskaičiuota išvestis sutaptų su Python modelio išvestimi.

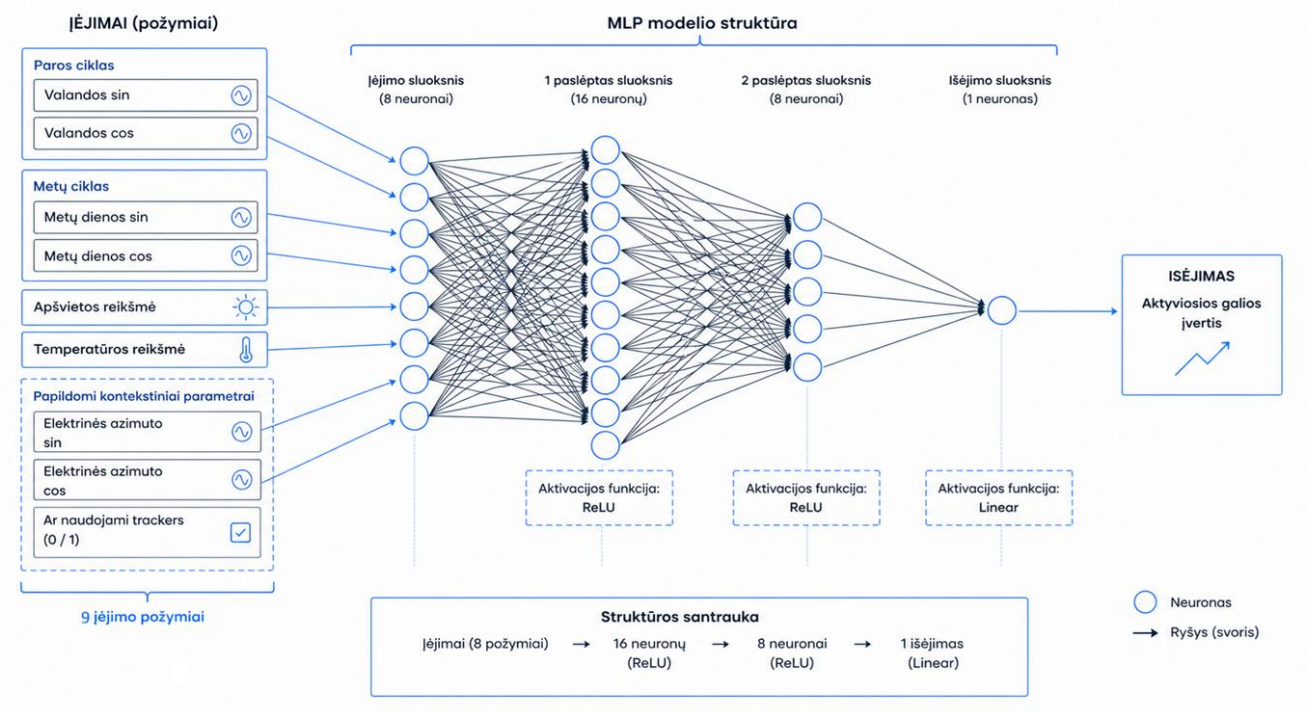
Decentralizuotos architektūros privalumas išryškėjo jau šiame etape. Python pusėje buvo galima greitai keisti požymius, filtravimo sąlygas, tinklo struktūrą, mokymo parametrus ir vertinimo metodus. Tuo tarpu PLC pusėje buvo kuriamas atskiras taikymo mechanizmas, kuris leido priimti apmokyto modelio parametrus ir realiuoju laiku skaičiuoti aktyviosios galios prognozę. Vėliau ši kryptis buvo išplėtota iki automatinio modelių mokymo ir perdavimo proceso.

3.2.1. Pirmojo MLP modelio sudarymas

Pirmajam praktiniam bandymui buvo pasirinktas MLP tipo neuroninis tinklas. Šis modelis buvo naudojamas kaip pradinis etalonas ir kaip bazė tolesnei PLC integracijai. Pirmoji MLP modelio versija buvo sudaryta iš įėjimo sluoksnio, dviejų paslėptų sluoksnių ir išėjimo sluoksnio.

Modelis naudojo 9 įėjimo kintamuosius, iš kurių pagrindiniai buvo apšvieta ir modulio temperatūra, o papildomi požymiai aprašė paros laiką, metų dieną, azimutą ir sekimo sistemų požymį. Įėjimų normalizavimui buvo naudotos vidurkio reikšmės. PLC realizacijoje buvo naudojami du paslėpti sluoksniai su 16 ir 8 neuronais, o galutinė prognozė apskaičiuojama kaip aktyvioji galia kW vienetais.

Python aplikacija šiuo etapu dar neturėjo pilnos vizualizavimo ir automatinio palyginimo infrastruktūros. Ji leido apmokyti MLP modelį, stebėti mokymo paklaidą ir naudoti funkcijas, kuriose pagal pasirinktą laiką, apšvietą, temperatūrą ar kitus požymius buvo apskaičiuojama modelio prognozė ir lyginama su realia tuo metu buvusia aktyviąja galia. Tokia minimalistinė struktūra buvo pasirinkta todėl, kad pirmasis tikslas buvo ne rezultatų vizualizavimas, o technologinės grandinės patikrinimas



5 pav. Pradinė MLP modelio struktūra

Modelio išėjimas buvo aktyviosios galios įvertis. Šiuo etapu svarbiausia buvo patikrinti, ar neuroninis tinklas geba bent dalinai atkartoti realios elektrinės generacijos dinamiką, ir ar jo skaičiavimus įmanoma perkelti į *CODESYS* ST kalbą. Pirminėje versijoje buvo naudojamas platesnis požymių rinkinys, į kurį pateko ne tik apšvieta ir temperatūra, bet ir laiko bei geometriniai požymiai, tokie kaip paros laikas, metų diena, azimutas, kampai ir kiti papildomi signalai. Tokia pradžia buvo logiška, nes literatūrinėje analizėje akcentuojama, kad PV galia priklauso nuo meteorologinių ir laiko veiksnių, o DI modeliai gali išmokti netiesines priklausomybes tarp daugelio įėjimų ir generacijos.

Vis dėlto praktiniai rezultatai parodė, kad platus požymių rinkinys ne visada pagerina modelio veikimą. Dalis įėjimų buvo statiški arba silpnai informatyvūs realaus laiko prognozavimo uždaviniui, o kai kurie signalai galėjo įnešti papildomo triukšmo. Pirminiame etape taip pat nebuvo pilnai išspręsta duomenų filtravimo problema. Mokymui buvo naudojamos eilutės, kuriose aktyvioji galia buvo didesnė nei 0, o apšvieta viršijo 50 W/m² ribą, tačiau papildomas filtravimas pagal ribojimo būsenas ar kitus elektrinės darbo režimus dar nebuvo pilnai realizuotas. Dėl to mokymo duomenyse galėjo likti situacijų, kai reali aktyvioji galia neatspindėjo maksimalaus galios potencialo, pavyzdžiui, dėl elektrinės ribojimo arba kitų eksploatacinių būsenų. Šis duomenų netolygumas ir pradinis požymių parinkimas turėjo tiesioginę įtaką modelio tikslumui.

Šiame etape gautas rezultatas buvo vertingas ne dėl galutinės paklaidos, o dėl to, kad buvo sukurta veikianti MLP mokymo ir eksportavimo grandinė. Modelis jau galėjo būti apmokomas naudojant „PyTorch“ biblioteką, jo svoriai ir poslinkiai galėjo būti ištraukiami iš Python aplinkos, o vėliau panaudojami PLC programoje.

3.2.2. Pirminė CODESYS ST realizacija

Sukūrus pirmąjį MLP modelį, buvo pereita prie svarbiausios praktinio pritaikymo fazės – modelio realizavimo CODESYS aplinkoje. Šio etapo tikslas buvo patikrinti, ar apmokyto neuroninio tinklo struktūrą galima perkelti į PLC valdiklį ir vykdyti realiuoju laiku. Kadangi buvo siekiama išlaikyti

pilną kontrolę ir suprasti modelio skaičiavimo eigą, buvo pasirinktas metodas modelio inferenciją *CODESYS* ST kalba realizuoti „nuo nulio“, nenaudojant anksčiau aptartos ML bibliotekos.

Toks sprendimas leido tiksliai valdyti, kaip PLC pusėje interpretuojami modelio įėjimai, svoriai, poslinkiai, aktyvacijos funkcijos ir išvesties denormalizavimas. MLP modelio skaičiavimas buvo išskaidytas į sluoksnius, kurių kiekviename atliekamas matricos ir įėjimo vektoriaus dauginimas, pridedamas poslinkis, o po to taikoma aktyvacijos funkcija. Tokia logika atitinka standartinę MLP skaičiavimo eigą, tačiau „CODESYS“ aplinkoje ją reikėjo realizuoti rankiniu būdu, naudojant masyvus, ciklus ir realiųjų skaičių operacijas.

Didelis dėmesys buvo skirtas tam, kad CODESYS aplinkoje apskaičiuota modelio išvestis kuo tiksliau sutaptų su „Python“ aplinkoje gauta išvestimi. Tai buvo daroma, nes Python ir PLC aplinkos skiriasi skaičių tipais, masyvų indeksavimu, duomenų paruošimo logika ir laiko požymių interpretavimu. Todėl buvo testuojami keli įėjimų apdorojimo ir svorių išdėstymo variantai. Reikėjo užtikrinti, kad Python pusėje eksportuoti parametrai CODESYS pusėje būtų nuskaitomi ta pačia tvarka, kokia jie buvo naudojami modelio struktūroje. Taip pat reikėjo teisingai atkartoti normalizavimo veiksmus, nes net nedidelis skirtumas tarp Python ir PLC įėjimo mastelių galėjo lemti skirtingą prognozuojamą galią **(žr. 1 priedas)**.

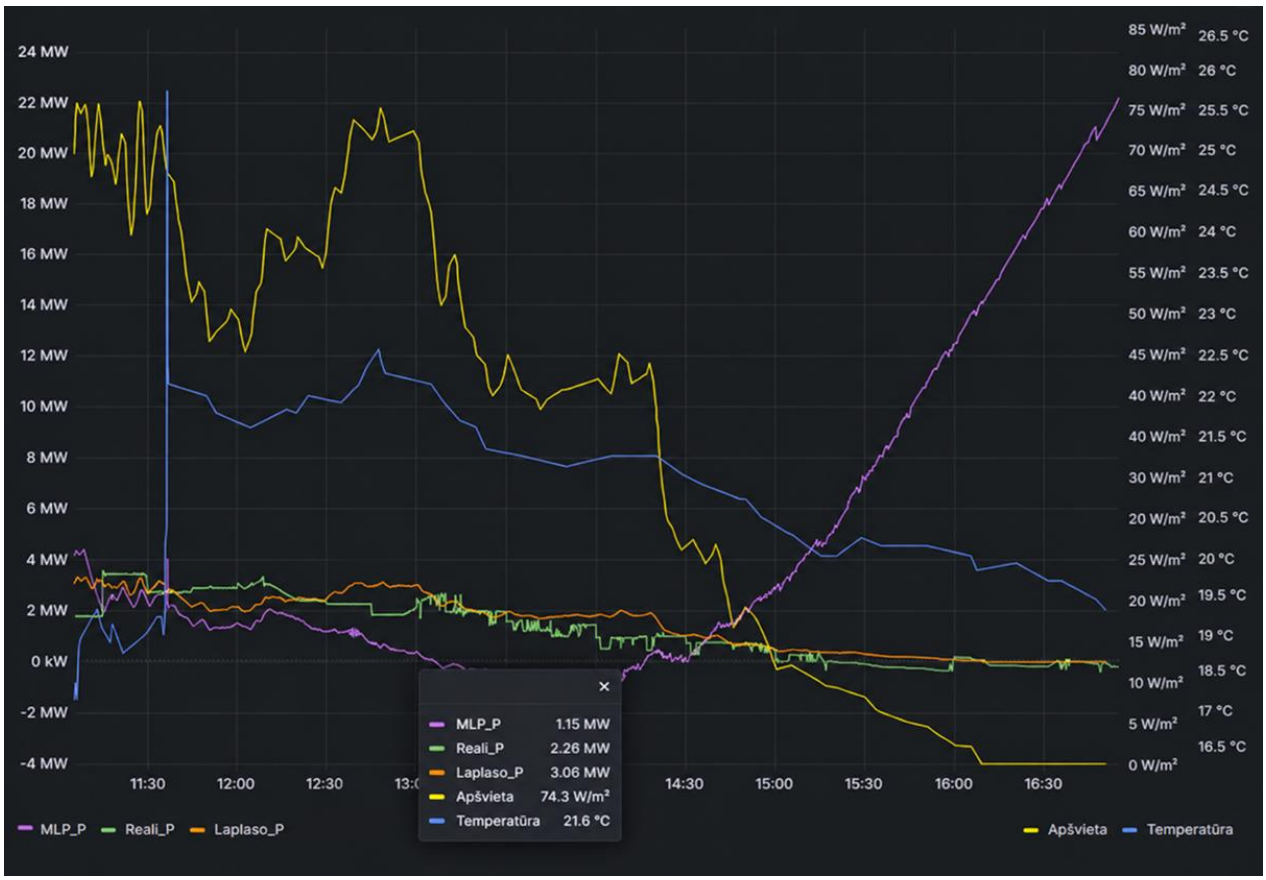
Atlikus kelias ST realizacijos iteracijas, buvo pasiekta struktūra, kurioje CODESYS programa teisingai interpretuodavo modelio sluoksnius ir grąžindavo praktiškai tą pačią reikšmę kaip Python modelis. Tai buvo svarbus tyrimo rezultatas, nes patvirtino, kad MLP tipo modelio inferencija gali būti realizuojama valdiklyje be papildomos ML bibliotekos, naudojant standartinę ST kalbos logiką.

3.2.3. Pirmosios iteracijos bandymas realioje elektrinėje

Sėkmingai perkėlus pirmąjį MLP modelį į CODESYS aplinką, buvo atliktas pirminis bandymas realioje elektrinės valdymo sistemoje. Pirmojo modelio prognozė realioje elektrinėje pradžioje sekė generacijos dinamiką, tačiau apšvietai krentant modelio išvestis „nuvažiavo į begalybę“ . Tai rodo, kad pirminė modelio versija buvo netinkama praktiniam naudojimui kaip galutinis sprendimas, tačiau ji patvirtino esminę technologinę hipotezę: Python aplinkoje apmokytas MLP modelis gali būti realizuotas PLC valdiklyje ir naudojamas realaus laiko signalų apdorojimo grandinėje.

2 lentelė. Pirmos iteracijos MLP modelio palyginimas su realios elektrinės Laplaso modeliu

Modelis	**Lyginama su**	**MAE, kW**	**RMSE, kW**
Laplaso modelis	Reali P galia	~500 - 1000	~800 - 1300
MLP neuroninis tinklas	Reali P galia	~4000 - 6000	~6000 - 8000



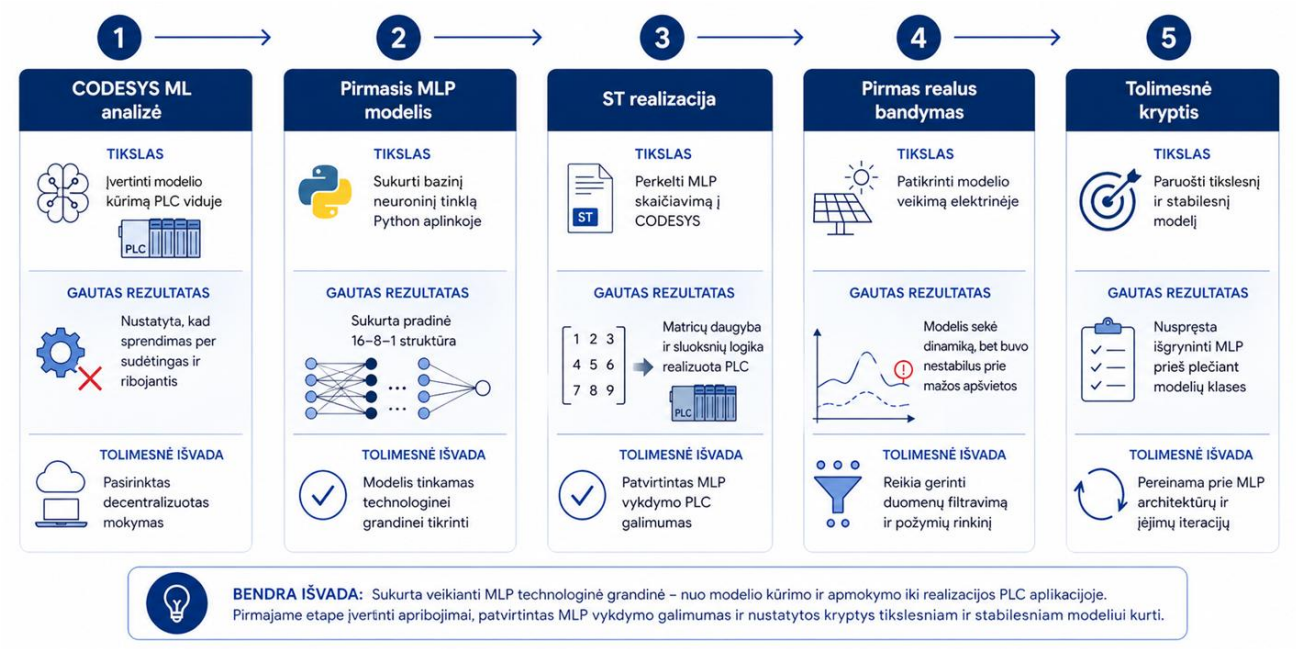
6 pav. Pirmojo modelio iteracijos prognoze PLC programoje palyginimas su tuometine Laplaso modelio ir realia aktyvia galia

Gauti rezultatai atskleidė kelias pagrindines problemas:
– modelis buvo per jautrus tam tikroms įėjimo reikšmėms;
– mokymo duomenų rinkinys nėra pakankamai gerai išfiltruotas;
– modelis neturėjo pakankamai fizinių ribojimų;
– pradinė požymių struktūra buvo per plati.

Šis etapas buvo svarbus tuo, kad jis aiškiai atskyrė dvi problemas: modelio matematinio perkėlimo į PLC problemą ir modelio tikslumo problemą. Matematinio perkėlimo problema buvo išspręsta – CODESYS aplinka galėjo vykdyti MLP struktūrą. Tačiau modelio tikslumo ir stabilumo problema liko atvira. Dėl to buvo nuspręsta tolimesnėje tyrimo eigoje ne plėsti modelių skaičių iš karto, o pirmiausia išgryninti MLP modelio mokymo logiką, požymių rinkinį, duomenų filtravimą ir architektūrą.

3.2.4. Pirmojo etapo išvada

Šis etapas tapo pagrindu tolimesnei tyrimo eigai. Toliau darbe bus analizuojama, kaip keičiant MLP įėjimo požymius, duomenų filtravimo taisykles ir tinklo struktūrą buvo siekiama pagerinti modelio tikslumą bei stabilumą. Tik sukūrus pakankamai patikimą MLP modelį, bus pereinama prie platesnio skirtingų modelių palyginimo bei jų tinkamumo PLC realizacijai vertinimo.



7 pav. Pirmojo tyrimo etapo eiga

3.3. MLP modelio stabilizavimas ir požymių rinkinio mažinimas

Po pirmojo eksperimentinio etapo buvo padaryta išvada, kad tolimesnis darbas turi būti nukreiptas ne į modelio sudėtingumo didinimą, o į jo stabilizavimą. Pirmoji MLP modelio versija patvirtino, kad Python aplinkoje apmokytą neuroninį tinklą galima perkelti į CODESYS ST aplinką ir vykdyti PLC valdiklyje, tačiau jo tikslumas ir stabilumas dar nebuvo pakankami praktiniam naudojimui. Dėl šios priežasties tolimesniame etape buvo siekiama sumažinti įėjimo požymių skaičių, atsisakyti silpnai informatyvių arba statiškų reikšmių ir sukurti patikimesnę modelių testavimo struktūrą.

Pirmosios iteracijos metu modelyje buvo naudojama daugiau įėjimo požymių, įskaitant apšvietą, temperatūrą, paros ir metų laiko požymius, azimutą bei kitus statinius parametrus. Tačiau bandymai parodė, kad dalis šių požymių nesuteikia pakankamai naudingos informacijos konkrečiai tiriamai elektrinei. Kadangi modeliai šiame darbe nėra kuriami kaip universalūs visoms elektrinėms, o apmokomi konkrečiam objektui, tokios reikšmės kaip azimutas ar panelių posvyrių kampai tampa pastovūs ir praranda reikšmę modelio mokymui. Tokie požymiai būtų reikšmingi tik tada, jei vienas modelis būtų mokomas kelioms skirtingoms elektrinėms arba jei elektrinėje būtų įrengti saulės sekikliai, keičiantys modulių padėtį laike. Tiriamoje elektrinėje sekikliai nenaudojami, todėl šiame etape buvo nuspręsta jų požymio nenaudoti. Todėl tolimesniame tyrimo etape įėjimo požymių rinkinys buvo sumažintas iki apšvietos ir ciklinių laiko požymių. Tokiu būdu modelis buvo mokomas pagal penkis įėjimus:

1. *irradiance* – saulės apšvieta;
2. *hour_sin* – paros laiko sinusinė dedamoji;
3. *hour_cos* – paros laiko kosinusinė dedamoji;
4. *doy_sin* – metų dienos sinusinė dedamoji;
5. *doy_cos* – metų dienos kosinusinė dedamoji.

3.3.1. Duomenų struktūros ir mokymo rinkinio koregavimas

Didelę įtaką modelio tikslumui turėjo ne tik neuroninio tinklo architektūra, bet ir naudojamų duomenų kokybė. Tyrimo metu buvo nustatyta, kad nuo duomenų kaupimo pradžios keitėsi duomenų bazės struktūra, signalų įrašinėjimo logika ir kai kurių matavimų formatai. Dėl to nuspręsta modelį mokyti pagal naujesnę ir stabilesnę duomenų struktūrą, kuri pradėta naudoti nuo 2024 m. liepos mėnesio.

Senesnėje duomenų struktūroje aktyvioji galia ne visada buvo saugoma kaip bendra elektrinės suminė generacija. Kai kuriais laikotarpiais ją reikėdavo rekonstruoti sumuojant atskirų keitiklių matavimus SQL užklausoje. Toks metodas kėlė papildomų paklaidų, nes atskiri keitiklių duomenys ne visada buvo registruojami tą pačią sekundę. Dėl skirtingo signalų atėjimo laiko, *null* reikšmių, ryšio sutrikimų ar atskirų įrenginių atsijungimų suminė galia galėjo būti apskaičiuojama netiksliai. Tai ypač svarbu neuroninio tinklo mokymui, nes modelis jautriai reaguoja į sistemines duomenų klaidas.

Pradiniuose bandymuose modelio tikslui buvo naudojami tinklo analizatoriaus aktyviosios galios matavimai. Tačiau vėliau ši logika buvo pakeista. Tinklo analizatoriaus matavimas prijungimo taške apima ne tik DC-DC keitiklių generaciją, bet ir elektrinės savąsias reikmes, transformatorių bei kabelių nuostolius. Kadangi šio darbo tikslas yra modeliuoti fotovoltinės elektrinės generuojamos galios potencialą, tikslingiau naudoti keitiklių generuojamos aktyviosios galios sumą. Naujesnėje duomenų struktūroje ši reikšmė sumuojama valdiklio cikle ir į duomenų bazę įrašoma kaip viena standartizuota elektrinės saulės generacijos reikšmė. Tai sumažina SQL užklausos metu atsirandančių sinchronizavimo klaidų tikimybę. Analogiška problema buvo pastebėta ir apšvietos matavimuose. Dalis apšvietos daviklių turėjo atsijungimų, o jų struktūra istorijos eigoje taip pat keitėsi. Kadangi apšvieta yra pagrindinis modelio įėjimas, netikslūs arba nevienodai suformuoti apšvietos duomenys tiesiogiai blogina modelio tikslumą. Todėl tolimesniame etape buvo nuspręsta naudoti stabilesnę naujos struktūros duomenų dalį, kurioje aktyvioji galia ir apšvieta jau buvo formuojamos vieningiau.

3.3.2. Modelio testavimo aplinkos sukūrimas

Norint sistemingai gerinti modelį, pirmiausia reikėjo sukurti atskirą testavimo aplinką Python programoje. Pirmosiose programos versijose modelio tikslumas buvo tikrinamas fragmentiškai, pagal pavienes reikšmes arba atskirus palyginimus. Tolimesniam darbui toks metodas nebuvo pakankamas, nes nebuvo galima objektyviai įvertinti, kaip modelis elgiasi visos dienos eigoje, kaip jis seka generacijos kreivę ir kokia yra bendra paklaida. Dėl šios priežasties Python programoje buvo pridėta atskira testavimo dalis, skirta pasirinktai parai analizuoti. Jos principas buvo toks: pagal pasirinktą datą iš duomenų bazės užklausiami tos paros duomenys, suformuojami modelio įėjimo požymiai, pritaikomas apmokytas MLP modelis ir gaunama prognozuojama aktyviosios galios kreivė visai parai. Kartu iš duomenų bazės grąžinama ir reali tuo metu buvusi aktyvioji galia, todėl galima atlikti tiek grafinį, tiek skaitinį palyginimą.

Pasirenkama testavimo data

```
•test_date = "2026-03-20"
```

MLP struktūra

```
•df_day = fetch_day_data_from_database(test_date)
```

Sudaromi modelio įėjimo požymiai ir normalizuojami

```
•X_day = build_features(df_day)
•X_day_scaled = x_scaler.transform(X_day)
```

Modelis pritaikomas pasirinktai dienai

```
•model.eval()
•with torch.no_grad():
•    y_pred_scaled = model(torch.from_numpy(X_day_scaled))
```

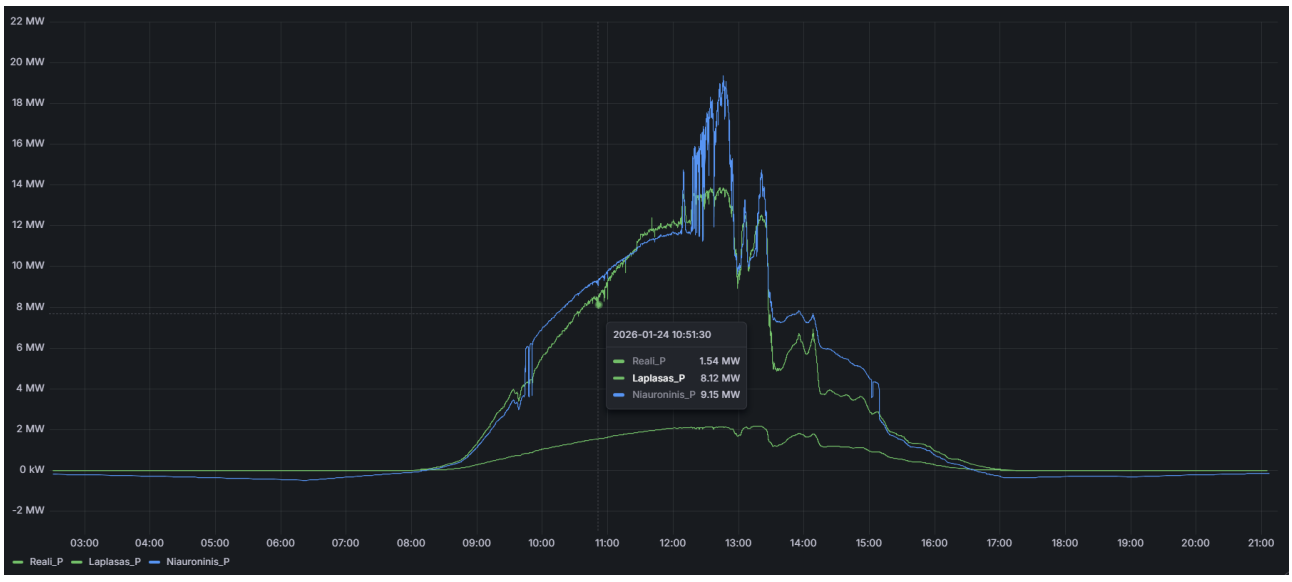
Prognozė grąžinama į fizikinę kW skalę ir palyginamas su relia aktyvia galia

```
•y_pred = y_scaler.inverse_transform(y_pred_scaled.numpy()
•y_true = df_day["ActivePower"].to_numpy()
•mae = np.mean(np.abs(y_pred - y_true))
•rmse = np.sqrt(np.mean((y_pred - y_true) ** 2))
```

8 pav. „Test_NN.py“ testavimo aplinkos kodo ištrauka

Toks testavimo principas leido vertinti modelį ne tik pagal bendrą mokymo paklaidą, bet ir pagal tai, kaip jis realiai atkartoja elektrinės generacijos kreivę. Tai svarbu, nes saulės elektrinės generacija per parą nėra stacionari – ryte ir vakare galia kinta staigiai, vidurdienį gali būti pasiekiamas pikas, o debesuotumas sukelia trumpalaikius šuolius. Todėl modelio kokybė buvo vertinama ne tik pagal MAE ar RMSE metrikas, bet ir vizualiai lyginant prognozuotą kreivę su realia aktyviąja galia.

Testavimo datų pasirinkimas taip pat tapo svarbia tyrimo dalimi. Jei testavimui pasirenkama diena, kurios metu elektrinė buvo ribojama, modelio tikslumo įvertinimas tampa nepatikimas. Tokiu atveju reali aktyvioji galia neatspindi maksimalaus galios potencialo, nes elektrinė generuoja ne tiek, kiek galėtų pagal meteorologines sąlygas, o tiek, kiek leidžia valdymo užduotis. Todėl testavimui buvo siekiama rinktis dienas, kuriose nėra aktyvaus galios ribojimo arba jo įtaka yra minimali.



9 pav. Pirmos iteracijos MLP modelio grafinis palyginimas su realios elektrinės Laplaso modeliu

3 lentelė. Pirmos iteracijos MLP modelio matematinis palyginimas su realios elektrinės Laplaso modeliu

Modelis	**Lyginama su**	**MAE, kW**	**RMSE, kW**
Laplaso modelis	Reali P galia	8464,62	8847,99
MLP neuroninis tinklas	Reali P galia	9277,63	9740,70

Apibendrinant, 2026-01-24 dienos grafikas rodo, kad patobulintas MLP modelis jau elgiasi stabiliau ir jo išvestis daugeliu laiko momentų supanašėja su Laplaso modelio kreive. Tai leidžia teigti, kad neuroninis tinklas, naudodamas apšvietos ir ciklinius laiko požymius, geba pakankamai gerai atkartoti bendrą elektrinės generacijos formą. Lyginant su ankstesniu modeliu (žr. 6 pav.), nebestebimas kritinis nestabilumas mažėjant apšvietai – modelio išvestis išlieka ribota ir fiziškai logiška, todėl galima teigti, kad požymių rinkinio sumažinimas ir duomenų filtravimo gerinimas turėjo teigiamą įtaką stabilumui. Tačiau kartu matyti, kad tiek Laplaso modelis, tiek MLP modelis vis dar pasižymi didele paklaida lyginant su realia aktyviąja galia. Abu modeliai nepakankamai tiksliai atkuria piko sritį ir neatsižvelgia į papildomus fizinius veiksnius, tokius kaip modulių temperatūra. Dėl šios priežasties modeliai pervertina generaciją. Vis dėlto, lyginant su pirmąja iteracija, pasiektas esminis progresas – modelis tapo stabilesnis, fiziškai interpretuojamas ir tinkamesnis tolimesniam tobulinimui. Tai pagrindžia sprendimą tolimesniuose etapuose koncentruotis į MLP modelio tikslumo gerinimą, papildant jį reikšmingesniais fizikiniais požymiais ir gerinant duomenų kokybę.

3.3.3. Neuroninio tinklo optimalios struktūros paieška

Parengus testavimo aplinką ir stabilizavus modelį, tolimesnis žingsnis buvo neuroninio tinklo struktūros paieška. Šiame etape nebuvo siekiama papildyti modelio naujais įėjimo duomenimis. Pirmiausia buvo norima nustatyti, kokia dviejų paslėptų sluoksnių MLP struktūra geriausiai tinka sumažintam požymių rinkiniui. Tam buvo keičiami neuronų skaičiai pirmajame ir antrajame paslėptuose sluoksniuose. Neuronų skaičius paslėptuose sluoksniuose tiesiogiai veikia modelio gebėjimą aprašyti netiesines priklausomybes. Per mažas neuronų skaičius gali lemti nepakankamą modelio lankstumą – toks modelis gali nesugebėti tiksliai atkartoti aktyviosios galios priklausomybės nuo apšvietos ir laiko. Per didelis neuronų skaičius, priešingai, gali lemti persimokymą, kai modelis pernelyg gerai prisitaiko prie mokymo duomenų, bet prasčiau veikia su naujais duomenimis. Modelio struktūra Python kode aprašyta kaip trijų pilnai sujungtų sluoksnių tinklas. Pirmuose dviejuose sluoksniuose naudojama ReLU aktyvacijos funkcija, o išėjimo sluoksnis yra tiesinis.

Modelio mokymui buvo naudojama MSE paklaidos funkcija ir Adam optimizavimo algoritmas. Taip pat buvo įdiegta ankstyvo stabdymo logika, kuri leidžia nutraukti mokymą tada, kai validavimo paklaida nebegerėja. Tai svarbu siekiant sumažinti persimokymo riziką.

Bazinė MLP struktūra, paieška atliekama keičiant fc1 ir fc2 sluoksnių neuronų skaičių.

```
class PowerNet(nn.Module):

    def __init__(self, input_dim: int):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 16)
        self.fc2 = nn.Linear(16, 8)
        self.fc3 = nn.Linear(8, 1)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

MLP treniravimas naudojant Adam algoritmą ir MSE paklaidos f-ją (validavimas)

```
criterion = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)

best_state = copy.deepcopy(model.state_dict())
best_val_rmse = float("inf")
patience_counter = 0

for epoch in range(EPOCHS):
    model.train()

    for xb, yb in train_loader:
        optimizer.zero_grad()
        preds = model(xb)
        loss = criterion(preds, yb)
        loss.backward()
        optimizer.step()

    model.eval()
    with torch.no_grad():
        val_preds = model(X_val_t)
        val_rmse = _scaled_rmse(val_preds, y_val_t)

    improved = (best_val_rmse - val_rmse) > EARLY_STOPPING_MIN_DELTA

    if improved:
        best_val_rmse = val_rmse
        best_state = copy.deepcopy(model.state_dict())
        patience_counter = 0
    else:
        patience_counter += 1

    if patience_counter >= EARLY_STOPPING_PATIENCE:
        break

model.load_state_dict(best_state)
```

10 pav. MLP modelio treniravimo Python kodo ištrauka

Ši logika užtikrina, kad galutiniam naudojimui išsaugomas ne paskutinės epochos modelis, o tas modelis, kuris validavimo duomenyse pasiekė geriausią rezultatą. Pagal atliktą architektūrų paiešką buvo patikrinti 78 skirtingi dviejų paslėptų sluoksnių MLP modeliai. Mokymui buvo naudojami apie 29,65 mln. eilučių. Pritaikius ribojimo filtrą pagal apšvietos ir galios gaubtinę (žr. 3.3.4 skyrių), mokymui buvo palikta apie 5,66 mln. eilučių.

4 lentelė. MLP architektūros paieškos be temperatūros rezultato santrauka

Modelio įėjimai	Architektūrų skaičius	Geriausia struktūra	MAE, kW	RMSE, kW
Apšvieta, paros laikas, metų diena	78	5 - 7 - 4 - 1	432,836	764,297

Šiame etape gautas rezultatas parodė, kad sumažintas požymių rinkinys ir tinkamai parinkta MLP architektūra leidžia gauti pakankamai stabilų modelį, kuris jau gali būti lyginamas su elektrinėje naudojamu Laplaso transformacijos pagrindu veikiančiu modeliu. Nors Laplaso modelis tam tikrais trumpais laikotarpiais gali būti tikslesnis, jis yra koreguojamas ir derinamas pagal konkrečias sąlygas. MLP modelis šiuo atveju mokomas iš ilgesnio istorinio laikotarpio, todėl jo tikslas yra ne idealiai atkartoti vienos dienos sukalibruotą modelį, o apibendrinti elektrinės elgseną skirtingomis sąlygomis.

3.3.4. Galios ribojimo aptikimas pagal apšvietos ir galios gaubtinę

Viena svarbiausių problemų mokant modelį buvo galios ribojimo aptikimas. Jei elektrinė tam tikru momentu yra ribojama, reali aktyvioji galia nebeatspindi maksimaliai galimos generacijos. Tokios eilutės netinka modelio mokymui, nes modelis pradėtų mokytis, kad esant tam tikrai apšvietai elektrinė natūraliai generuoja mažesnę galią, nors realiai ji buvo sumažinta valdymo algoritmu. Vien tik aktyviosios galios užduoties *(angl. setpoint)* reikšmės nepakanka ribojimui patikimai nustatyti. Elektrinė gali nepasiekti užduoties reikšmės ir dėl kitų priežasčių, galia gali svyruoti apie užduotį dėl PID reguliatoriaus darbo, taip pat gali būti taikomos rampos (tolygus užduoties kitimas), kurios nėra tiesiogiai matomos paprastoje ribojimo būsenoje. Todėl buvo pasirinktas statistinis metodas, pagrįstas apšvietos ir aktyviosios galios viršutine gaubtine *(angl. upper envelope)*. Tokiu metodu kiekvienam apšvietos lygiui iš duomenų nustatoma didžiausia tipinė aktyvioji galia, atitinkanti neribotą elektrinės darbą. Duomenų taškai, kurių aktyvioji galia reikšmingai mažesnė už šią ribą esant tai pačiai apšvietai, klasifikuojami kaip ribojimo būsenos. Taip ribojimas identifikuojamas remiantis statistiniu galios nuokrypiu nuo maksimaliai pasiekiamos generacijos, o ne tiesioginiais valdymo signalais.

Tokia funkcija pirmiausia suskirsto duomenis į apšvietos intervalus, pavyzdžiui, kas 25 W/m². Toliau kiekviename apšvietos intervale funkcija ieško, kokią didžiausią galią elektrinė įprastai pasiekia. Tam nenaudojama absoliuti maksimali reikšmė, nes ji gali būti atsitiktinis šuolis arba matavimo klaida. Vietoje to naudojamas 0,99 kvantilis, t. y. reikšmė, už kurią mažesni yra 99 % to intervalo galios taškų. Pavyzdžiui, jei intervale 500–525 W/m² yra 100 matavimų, funkcija paima ne patį didžiausią tašką, o maždaug 98-ą pagal dydį reikšmę. Tarkime, ji lygi 8500 kW. Tuomet laikoma, kad prie tokios apšvietos elektrinė neribotomis sąlygomis gali pasiekti apie 8500 kW.

5 lentelė. Viršutinės gaubtinės vertinimo pavyzdys

Apšvieta	Reali P galia, kw	Gaubtinė galia, kw	Santykis	Išvada
500 W/m²	8200	8500	0,96	neribota
500 W/m²	6000	8500	0,71	ribota
500 W/m²	3000	8500	0,35	ribota

Jei santykis yra mažesnis nei 0,95, funkcija pažymi eilutę kaip tikėtinai ribotą. Kitaip tariant, jei elektrinė generuoja mažiau nei 95 % tos galios, kurią paprastai gali pasiekti prie tokios apšvietos, laikoma, kad tuo metu jos galia greičiausiai buvo dirbtinai apribota.

Filtro parametrai

```
def apply_envelope_curtailment_filter(
    df: pd.DataFrame,
    *,
    irr_bin_w: float = 25.0,                # apšvietos intervalų dydis (W/m²)
    min_irradiance: float = 50.0,           # minimalus apšvietos slenkstis analizei
    envelope_quantile: float = 0.99,        # viršutinės gaubtinės kvantilis (pvz. 99%)
    curtailment_ratio_threshold: float = 0.95, # riba ribojimui nustatyti
    min_points_per_bin: int = 30,           # minimalus taškų kiekis intervale
) -> pd.DataFrame:

    out = df.copy()
```

Pašalinamos eilutės su trūkstamais duomenimis

```
out = out.dropna(subset=["timestamp", "Irradiance", "ActivePower"])
    out = out[(out["Irradiance"] >= 0) & (out["ActivePower"] >= 0)]
    valid_env = out[out["Irradiance"] >= min_irradiance].copy()
```

Suskirstoma apšvieta į intervalus, skaičiuojamas kvantilis

```
valid_env["irr_bin"] = (
        np.floor(valid_env["Irradiance"] / irr_bin_w) * irr_bin_w
    ).astype(float)

    grouped = valid_env.groupby("irr_bin")["ActivePower"].agg(
        envelope_power=lambda s: s.quantile(envelope_quantile),
        n_points="count",
    ).reset_index()

    grouped = grouped[grouped["n_points"] >= min_points_per_bin].copy()
    grouped = grouped.sort_values("irr_bin")

    out["irr_bin"] = (
        np.floor(out["Irradiance"] / irr_bin_w) * irr_bin_w
    ).astype(float)
```

Sugrupuojami duomenys ir interpoliuojami pagal gaubtinę

```
x_env = grouped["irr_bin"].to_numpy(dtype=float)
    y_env = grouped["envelope_power"].to_numpy(dtype=float)

    out["envelope_power"] = np.nan
    interp_mask = out["Irradiance"] >= min_irradiance

    out.loc[interp_mask, "envelope_power"] = np.interp(
        out.loc[interp_mask, "Irradiance"].to_numpy(dtype=float),
        x_env,
        y_env,
        left=np.nan,
        right=np.nan,
    )
```

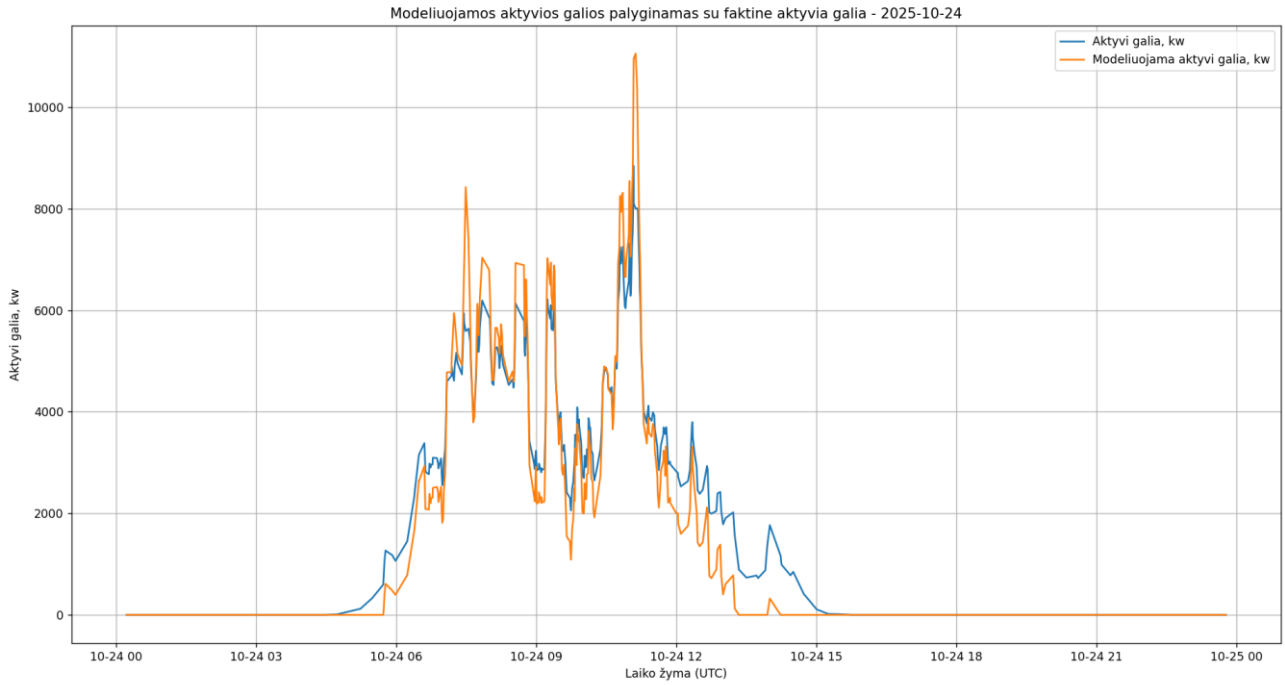
Apskaičiuojamas santykis ir aptinkamas ribojimas

```
out["power_envelope_ratio"] = out["ActivePower"] / out["envelope_power"]

    out["is_curtailed"] = False

    curtailed_mask = (
        (out["Irradiance"] >= min_irradiance)
        & out["envelope_power"].notna()
        & (out["power_envelope_ratio"] < curtailment_ratio_threshold)
    )

    out.loc[curtailed_mask, "is_curtailed"] = True

    return out
```

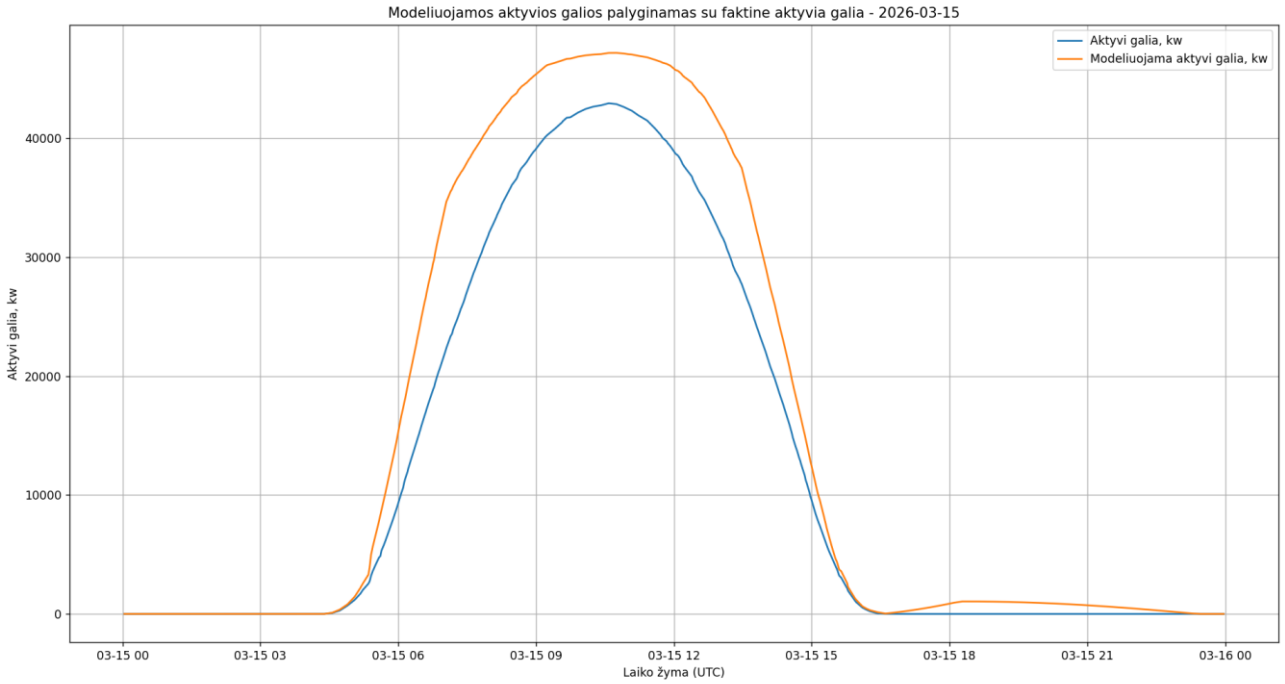
11 pav. Filtravimo funkcijos Python kodo ištrauka

Šis metodas taip pat reikšmingai pagerino mokymo duomenų kokybę, nes leido pašalinti dalį situacijų, kuriose elektrinė negeneravo maksimalios galimos galios. Vis dėlto tyrimo metu buvo

pastebėtas ir metodo taikymo apribojimas. Pradinėje realizacijoje ribojimo filtras buvo taikomas dar „žaliems“ duomenims, kuriuose aktyviosios galios ir apšvietos reikšmės ne visada sutapdavo pagal tą pačią sekundę. Dėl to pritaikius *dropna* operaciją galėjo būti pašalinama labai didelė dalis eilučių. Ši problema vėliau buvo sprendžiama keičiant duomenų sulyginimo ir apjungimo logiką, tačiau šiame etape ji turėjo įtakos modelių mokymo duomenų imčiai ir rezultatų interpretacijai.



12 pav. Šaltojo laikotarpio realios generacijos palyginimas su MLP be temperatūros (žr. 4 lentelę) prognoze



13 pav. Šiltojo laikotarpio realios generacijos palyginimas su MLP be temperatūros (žr. 4 lentelę) prognoze

Analizuojant grafikus galima daryti išvadą, kad pritaikius galios ribojimo aptikimo metodą ir išgryninus modelio struktūrą, pasiektas aiškus modelio veikimo pagerėjimas skirtingais sezonais.

Lyginant su ankstesnėmis iteracijomis, modelio prognozė tapo stabilesnė ir geriau atitinka realios generacijos formą tiek pereinamuoju laikotarpiu (spalio mėn.), tiek pavasario sąlygomis (kovo mėn.).

3.3.5. Temperatūros požymio įtraukimas į modelį

Kai buvo gautas pakankamai stabilus MLP modelis, naudojantis apšvietą ir laiko požymius, tolimesnis žingsnis buvo temperatūros įtraukimas. Pradiniame bandyme buvo svarstyta naudoti agreguotą visos elektrinės temperatūros reikšmę, kuri naudojama esamame Laplaso transformacijos pagrindu veikiančiame modelyje. Tačiau analizuojant duomenų bazės grąžinamus duomenis buvo pastebėta, kad laikui bėgant keitėsi tai, kokie konkretūs jutikliai įtraukiami į šią agreguotą temperatūrą analogiškai. Vienais laikotarpiais galėjo būti naudojami modulių temperatūros jutikliai, kitais – aplinkos temperatūros jutikliai, taip pat keitėsi jutiklių skaičius ir jų konfigūracija. (žr. 3.3.1 skyrių)

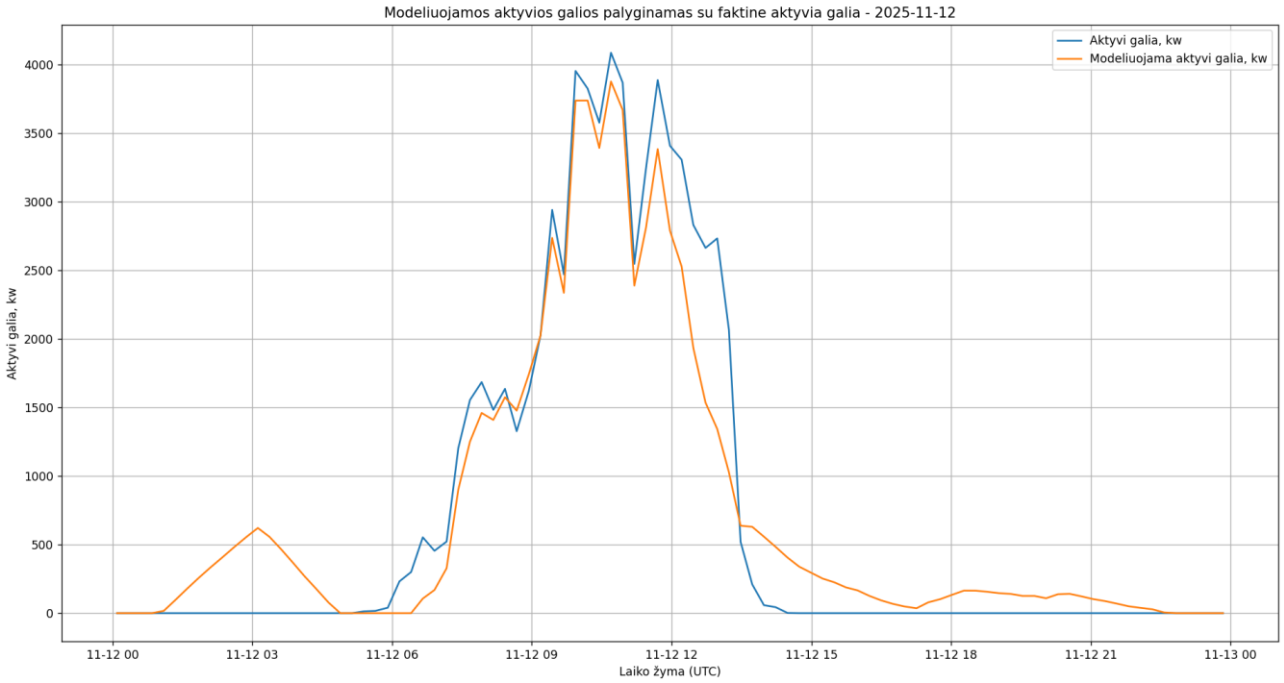
Dėl šios priežasties nuspręsta nenaudoti bendros agreguotos elektrinės temperatūros reikšmės kaip modelio įėjimo. Siekiant išvengti neteisingų arba nevienodai interpretuojamų duomenų, buvo pasirinkta naudoti vieną stabilų aplinkos temperatūros jutiklių, kurio konfigūracija nesikeitė visu pasirinktu mokymo laikotarpiu. Tokiu būdu modelio įėjimų rinkinys buvo papildytas temperatūra. tokio požymio įtraukimas reiškė, kad modelis galėjo mokytis ne tik priklausomybės tarp apšvietos, paros / metų laiko ir aktyviosios galios, bet ir temperatūros įtakos generacijos efektyvumui. Teoriškai toks modelis turėtų geriau atskirti situacijas, kai esant panašiai apšvietai generacija skiriasi dėl skirtingos aplinkos ar modulių temperatūros.

Vis dėlto šiame konkrečiame bandyme temperatūros įtraukimas iš karto nepagerino galutinės testavimo paklaidos. Modelio treniravimo metu buvo pastebėta, kad po duomenų filtravimo liko tik 33 920 neribotų eilučių, nors pradinis duomenų rinkinys turėjo apie 19,19 mln. eilučių. Taip nutiko dėl griežto duomenų apjungimo, *dropna* operacijos ir ribojimo filtro taikymo duomenims, kuriuose aktyviosios galios, apšvietos ir temperatūros reikšmės ne visada buvo registruotos tuo pačiu laiko momentu. Dėl to modelio mokymo imtis tapo per maža ir nepakankamai reprezentatyvi.

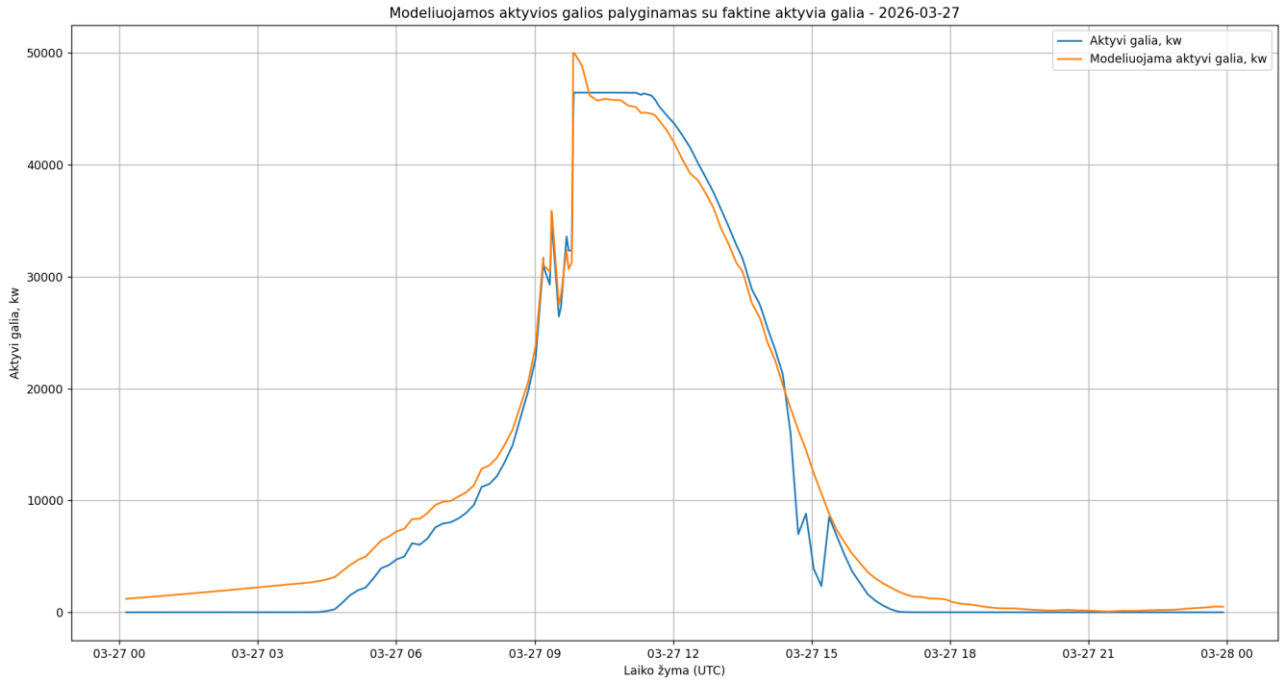
6 lentelė. MLP modelių be temperatūros ir su temperatūra palyginimas

Modelio įėjimai	Geriausia struktūra	Apmokymo eilutės	MAE, kW	RMSE, kW	Pastaba
Apšvieta, paros laikas, metų diena	5-7-4-1	5 657 226	432,836	764,297	Stabilus modelis su pakankama mokymo imtimi
Apšvieta, paros laikas, metų diena, temperatūra	6-8-4-1	33 920	1130,906	1743,901	Rezultatas pablogėjo dėl per mažos ir nepakankamai reprezentatyvios imties

Šie rezultatai nereiškia, kad temperatūra nėra reikšmingas požymis. Priešingai, fizikinė PV modulių veikimo logika rodo, kad temperatūra turi įtakos generuojamai galiai. Tačiau šis eksperimentas parodė, kad papildomo požymio įtraukimas pagerina modelį tik tada, kai pats požymis yra patikimas, stabiliai registruojamas ir teisingai sulygintas su kitais signalais. Todėl temperatūros modelio bandymas tapo svarbia išvada apie duomenų paruošimo reikšmę.



14 pav. Šaltojo laikotarpio realios generacijos palyginimas su MLP su temperatūra (žr. 6 lentelę) prognoze



15 pav. Šiltojo laikotarpio realios generacijos palyginimas su MLP su temperatūra (žr. 6 lentelę) prognoze

Vis dėlto temperatūros požymio vertinimas negali būti atliekamas tik pagal bendrą modelio treniravimo rezultatą. Nors bendroje apmokymo ir testavimo imtyje modelis su temperatūros požymiu parodė didesnę paklaidą nei modelis be temperatūros, atskirų panašių dienų analizė parodė, kad temperatūros požymis gali reikšmingai pagerinti prognozavimo tikslumą. Tai reiškia, kad bendras paklaidos padidėjimas buvo labiau susijęs ne su pačios temperatūros kaip požymio nenaudingumu, o su tuo, kad dėl griežtesnio duomenų apjungimo buvo prarasta didelė dalis mokymo eilučių ir sumažėjo mokymo imties reprezentatyvumas.

Pritaikius modelį konkrečioms dienoms, temperatūros požymio įtraukimas pagerino prognozavimo tikslumą tiek šiltesnio, tiek šaltesnio laikotarpio sąlygomis. (žr. 7 lentelę)

7 lentelė. MLP modelio be temperatūros ir su temperatūros požymiu palyginimas panašiomis dienomis

Sezoninis atvejis	Modelis su temperatūra	Testavimo data	MSE	RMSE, kW	MAE, kW
Šiltasis	Taip	2026-03-27	4 511 745,90	2 124,09	1 592,20
	Ne	2026-03-15	44 522 904,45	6 672,55	5 481,97
Šaltasis	Taip	2025-11-12	116 890,27	341,89	228,07
	Ne	2025-10-24	466 122,79	682,73	538,92

Pastaba: lentelėje palyginamos ne tos pačios datos, o panašios sezoninės dienos.

Apibendrinant galima teigti, kad temperatūros požymio įtraukimas buvo pagrįstas. Nors dėl griežto duomenų apjungimo sumažėjo mokymo duomenų kiekis, atskirų šaltojo ir šiltojo laikotarpio dienų palyginimas parodė, kad modelis su temperatūra tiksliau atkartoja realios generacijos kreivę. Tai leidžia daryti išvadą, kad temperatūros suteikiama fizikinė informacija buvo reikšmingesnė už dėl filtravimo prarastą mokymo imtį. Todėl tolimesnėse modelio iteracijose temperatūra turi būti paliekama kaip vienas pagrindinių įėjimo požymių, kartu gerinant duomenų sinchronizavimo logiką, kad būtų išsaugota kuo didesnė mokymo duomenų dalis.

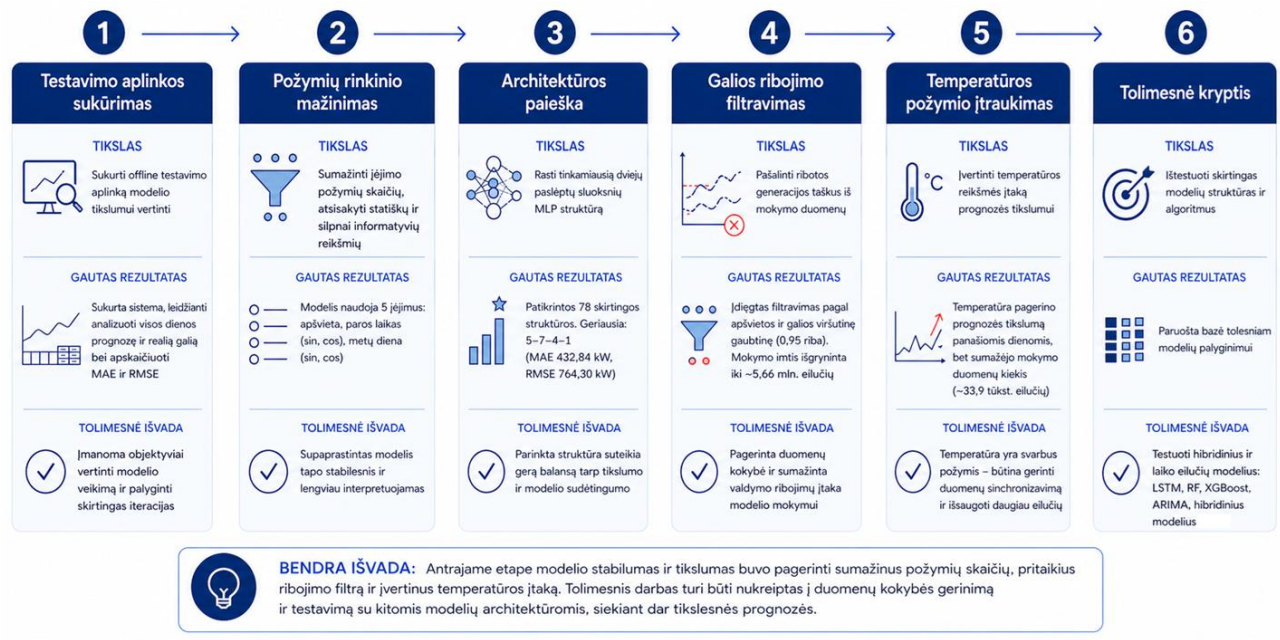
3.3.6. Antrojo tyrimo etapo apibendrinimas

Šiame tyrimo etape buvo pereita nuo pirminio technologinio prototipo prie sistemingo modelio stabilizavimo. Pirmiausia buvo sukurta atskira Python testavimo aplinka, leidžianti pasirinktai parai palyginti modelio prognozę su realia aktyviąja galia ir apskaičiuoti MAE bei RMSE paklaidas. Tada buvo sumažintas modelio įėjimo požymių skaičius, atsisakant statiškų elektrinės parametrų ir paliekant tik apšvietą bei ciklinius laiko požymius. Toks supaprastinimas leido sumažinti perteklinį triukšmą ir pasiruošti bazinį modelį tolimesniam tobulinimui.

Taip pat svarbu pabrėžti, kad nebuvo formuojami papildomi dinaminiai požymiai, tokie kaip apšvietos slankiojo lango vidurkis, standartinis nuokrypis, kitimo greitis ar kelių elektrinės zonų tarpusavio skirtumai. Dėl to debesuotumo dinamika į modelius patenka tik netiesiogiai per momentinę apšvietos reikšmę ir laiko požymius. Tai riboja modelio gebėjimą prognozuoti staigius galios pokyčius kintamo debesuotumo sąlygomis.

Toliau buvo atlikta MLP architektūrų paieška, keičiant dviejų paslėptų sluoksnių neuronų skaičių. Taip pat buvo realizuotas ribojimo aptikimas pagal apšvietos ir galios viršutinę gaubtinę, kuris leido pašalinti dalį ribotos generacijos taškų iš mokymo rinkinio. Galiausiai įtrauktas temperatūros požymis, tačiau nustatyta, kad modelio treniravimo etapui yra būtina itin griežta duomenų kokybės kontrolė.

Šis etapas parodė, kad modelio tikslumas priklauso nuo trijų tarpusavyje susijusių veiksnių: teisingai parinktų įėjimo požymių, kokybiškai paruoštų mokymo duomenų ir tinkamai parinktos neuroninio tinklo struktūros. Gavus stabilų MLP modelį, tolimesniame tyrimo etape galima pereiti prie platesnio modelių palyginimo ir vertinti, ar kiti algoritmai, tokie kaip RF, ARIMA, LSTM ar hibridiniai modeliai, gali pasiekti geresnį tikslumą išlaikant praktinio realizavimo PLC aplinkoje galimybę.



16 pav. Antrojo tyrimo etapo eiga

3.4. Daugelio prognozavimo modelių palyginimas ir pasiruošimas automatiniam perdavimui į PLC

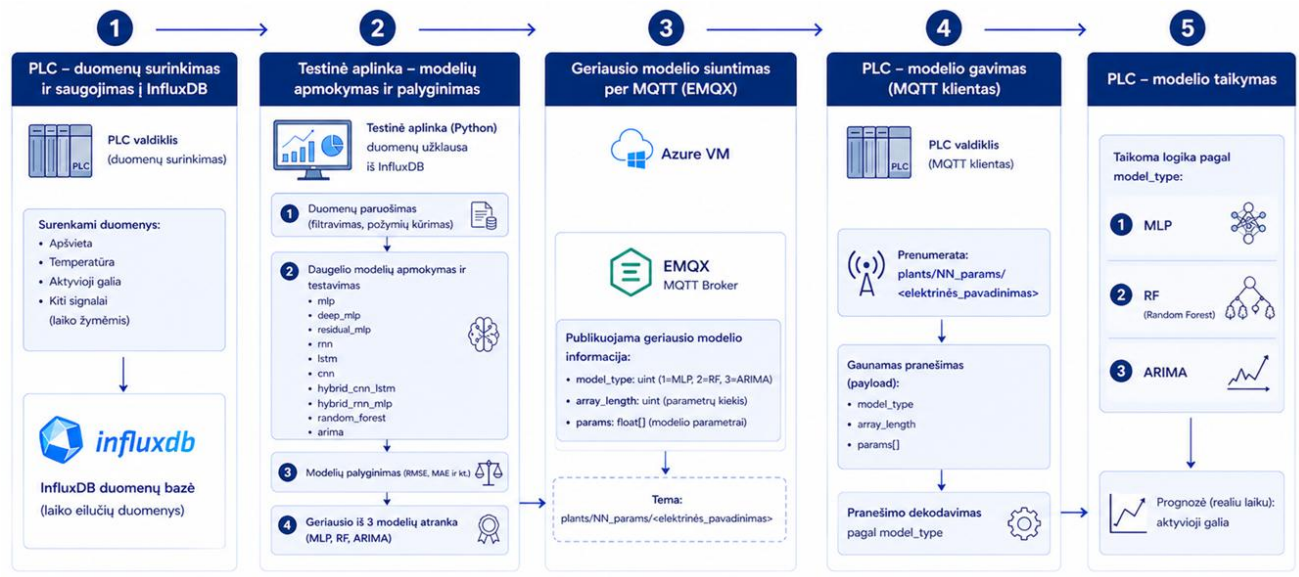
Po MLP modelio stabilizavimo ir temperatūros požymio įtraukimo buvo pereita prie platesnio tyrimo etapo – skirtingų prognozavimo modelių palyginimo. Ankstesniuose etapuose buvo siekiama išgryninti įėjimo požymius, sutvarkyti duomenų filtravimo logiką ir įsitikinti, kad neuroninio tinklo modelis gali būti realizuojamas PLC aplinkoje. Šiame etape pagrindinis tikslas buvo patikrinti, ar kiti modelių tipai gali pasiekti geresnį tikslumą už MLP ir ar jie gali būti praktiškai perduodami bei vykdomi pramoniniame valdiklyje. Šis tyrimo etapas buvo svarbus dėl dviejų priežasčių:

1. vien MLP modelio gerinimas neleidžia objektyviai įvertinti, ar pasirinktas metodas yra geriausias;
2. darbo tikslas nėra tik rasti mažiausią paklaidą Python aplinkoje – modelis turi būti tinkamas realiai eksploatacijai, t. y. jo parametrus turi būti įmanoma perduoti į PLC, o skaičiavimo logiką realizuoti CODESYS ST kalba.

Dėl šių priežasčių šiame etape buvo vertinami ne tik modelių tikslumo rodikliai, bet ir praktinio diegimo sudėtingumas.

3.4.1. Automatinės modelių mokymo grandinės sudarymas

Norint lyginti daug skirtingų modelių, ankstesnė vieno MLP modelio mokymo struktūra buvo išplėsta į bendrą modelių apmokymo grandinę. Tai leido automatiškai paimti istorinius duomenis iš „InfluxDB“ duomenų bazės, pritaikyti ribojimo filtrą, suformuoti įėjimo požymius, apmokyti kelis skirtingus modelius, apskaičiuoti jų paklaidas ir išsaugoti modelių parametrus tolimesniam eksportui. Pagrindinis šios logikos principas pateiktas 17 pav.



17 pav. Automatinė modelio mokymo, atrankos ir perdavimo į PLC struktūra

Šiame paveiksle vaizduojama, kaip PLC kaupia elektrinės, Python testavimo aplinka šiuos duomenis naudoja modelių mokymui, o geriausio modelio parametrai per MQTT brokerį perduodami atgal į PLC valdiklį. Tokia schema leidžia aiškiai atskirti tris pagrindines sistemos dalis: duomenų kaupimą, modelio mokymą ir modelio taikymą realiuoju laiku.

Duomenų surinkimo iniciavimas

```
df = fetch_pv_training_data_backwards(
    bucket=INFLUX_BUCKET,
    chunk_days=7,
    max_empty_chunks=3
)
```

Ribojimo filtro pritaikymas

```
if USE_ENVELOPE_CURTAILMENT_FILTER:
    df = apply_envelope_curtailment_filter(
        df,
        irr_bin_w=ENVELOPE_IRR_BIN_W,
        min_irradiance=ENVELOPE_MIN_IRRADIANCE,
        envelope_quantile=ENVELOPE_QUANTILE,
        curtailment_ratio_threshold=CURTAILMENT_RATIO_THRESHOLD,
        min_points_per_bin=ENVELOPE_MIN_POINTS_PER_BIN,
    )
    df = df[~df["is_curtailed"]].copy()
```

Modelių apmokymas, palyginimas, eksportas

```
trained_models = train_all_models(X_train, y_train, X_val, y_val)
metrics = evaluate_models(trained_models, X_test, y_test, y_scaler)
export_models(trained_models, feature_cols, x_scaler, y_scaler)
```

18 pav. Bendrinė „Python“ kodo seka geriausio modelio atrinkimui ir eksportui

Ši kodo dalis parodo, kad modelių palyginimas buvo atliekamas ne rankiniu būdu, o per bendrą automatizuotą grandinę. Visi modeliai buvo mokomi naudojant tą pačią paruoštą duomenų imtį, todėl jų paklaidos galėjo būti lyginamos tarpusavyje. Tai svarbu, nes modelių palyginimas būtų neobjektyvus, jei vienas modelis būtų mokomas su vienokiais duomenimis, o kitas su kitokiais.

3.4.2. Lyginti prognozavimo modeliai

Šiame tyrimo etape buvo lyginamos kelios modelių grupės: klasikinis MLP, gilesnis MLP, liekamojo tipo MLP *(angl. residual MLP)*, sekų modeliai, CNN, hibridiniai modeliai, atsitiktinių miškų modelis ir ARIMA. Toks modelių rinkinys pasirinktas tam, kad būtų galima palyginti skirtingus prognozavimo principus.

Svarbu pabrėžti, kad CNN modelis šiame darbe nebuvo taikomas vaizdų analizei (kam jis yra naudojamas įprastai). Jis buvo adaptuotas regresijos uždaviniui, naudojant vienmatę laiko eilučių struktūrą (1D). Modelio įėjimui buvo formuojamas požymių langas, kuriame pateikiamos kelių ankstesnių laiko momentų apšvietos, temperatūros ir laiko požymių reikšmės. Konvoliuciniai sluoksniai tokiu atveju naudojami ne erdviniams vaizdo požymiams, o lokaliems laiko dinamikos šablonams išskirti. Galutinis CNN modelio sluoksnis buvo tiesinis regresijos sluoksnis, kurio išėjimas yra prognozuojama aktyvioji galia kW. Tokiu būdu CNN šiame darbe buvo naudojamas kaip eksperimentinis laiko eilučių regresijos modelis, o ne kaip klasikinis vaizdų apdorojimo tinklas. Jo paskirtis buvo patikrinti, ar konvoliuciniai filtrai gali aptikti trumpalaikius įėjimo signalų dinamikos šablonus.

8 lentelė. Pagrindiniai šiame etape lyginti modeliai

Modelis (angl.)	**Veikimo principas**	**Vertinimo tikslas**
MLP	Pilnai sujungtas neuroninis tinklas	Bazinis kompaktiškas neuroninis modelis
Deep MLP	Gilesnis daugiasluoksnis perceptronas	Įvertinti, ar gilesnė struktūra pagerina tikslumą
Residual MLP	MLP su liekamosiomis jungtimis	Patikrinti, ar papildomos jungtys stabilizuoja mokymą
RNN	Rekursinis neuroninis tinklas	Įvertinti trumpalaikę sekos priklausomybę
LSTM	Ilgalaikės ir trumpalaikės atminties tinklas	Patikrinti, ar atminties mechanizmas pagerina prognozę
CNN	Konvoliucinis tinklas	Įvertinti lokalių laiko struktūrų aptikimą
Hybrid CNN-LSTM	CNN ir LSTM derinys	Sujungti lokalių požymių ir sekos atminties privalumus
Hybrid RNN-MLP	RNN ir MLP derinys	Sujungti sekos ir statinių požymių informaciją
Random Forest	Sprendimų medžių ansamblis	Patikrinti klasikinio ML modelio tikslumą
ARIMA	Autoregresinis laiko eilučių modelis	Įvertinti statistinio modelio tinkamumą

Kodo struktūroje kiekvienas modelių tipas buvo realizuotas kaip atskira šeima. MLP modelis išliko bazinis kandidatas, tačiau prie jo buvo pridėti sekų modeliai ir klasikiniai mašininio mokymosi metodai. Sekų modeliams buvo sukurta atskira duomenų paruošimo logika, kuri iš paprasto duomenų masyvo sudaro laiko sekas. Tai reikalinga RNN, LSTM, CNN ir hibridiniams modeliams, nes jie apdoroja ne vieną laiko tašką, o kelių ankstesnių taškų seką.

Duomenų paruošimas sekų modeliams

```
def _create_sequence_dataset(X: np.ndarray, y: np.ndarray, seq_len: int):
    X_seq = []
    y_seq = []

    for i in range(seq_len - 1, len(X)):
        X_seq.append(X[i - seq_len + 1:i + 1])
        y_seq.append(y[i])

    return np.asarray(X_seq, dtype=np.float32), np.asarray(y_seq,
dtype=np.float32)
```

19 pav. Python kodo ištrauka duomenų rinkinio paruošimui sekų modeliams

Tokiu būdu sekų modeliai galėjo įvertinti ne tik momentinę apšvietą, temperatūrą ir laiko požymius, bet ir jų pokytį keliuose ankstesniuose matavimo taškuose. Teoriškai tai turėtų padėti debesuotumo, staigių apšvietos pokyčių ar elektrinės dinamikos atvejais. Tačiau toks modelis tampa sudėtingesnis PLC realizacijai, nes reikia saugoti istorinius įėjimo taškus ir atlikti daugiau tarpinių skaičiavimų.

3.4.3. Modelių tikslumo palyginimas

Atlikus modelių apmokymą, visi modeliai buvo įvertinti pagal testavimo imtį. Pagrindinės vertinimo metrikos buvo RMSE ir MAE, apskaičiuotos kW vienetais. Kadangi saulės elektrinės galios prognozavime svarbios tiek vidutinės, tiek didelės momentinės paklaidos, naudotos abi metrikos.

Svarbu pabrėžti, kad šiame etape geriausio modelio atrinkimas buvo atliekamas remiantis matematiniais kriterijais, o ne grafine analize. Ankstesniuose etapuose *Test_NN* aplinkoje grafinis atvaizdavimas buvo naudojamas kaip svarbus įrankis modelių elgsenai įvertinti. Jis leido vizualiai palyginti prognozuojamą ir realią generacijos kreives bei identifikuoti akivaizdžias problemas. Nors techniškai *Test_NN* aplinka ir toliau leidžia vizualiai palyginti skirtingus modelius, šiame etape toks vertinimas jau netenka praktinės prasmės infrastruktūriniu požiūriu. Sukūrus pilną veikiančią struktūrą.

Tai reiškia, kad galutiniame etape modelio pasirinkimas grindžiamas tik objektyviomis paklaidų metrikomis, nes modelis jau yra integruotas į realaus laiko sistemą ir veikia kaip automatizuotos grandinės dalis.

9 lentelė. Daugelio modelių testavimo rezultatų palyginimas

Modelis (angl.)	**RMSE, kW**	**MAE, kW**	**Vertinimas**
Random Forest	523,269	124,241	Tiksliausias pagal testavimo metrikas
LSTM	864,279	368,499	Geras tikslumas, bet sudėtinga PLC realizacija
Hybrid RNN-MLP	1119,543	850,040	Vidutinis tikslumas, sudėtingesnė struktūra
Residual MLP	1284,505	836,487	Geresnis RMSE nei bazinis MLP, bet didesnis sudėtingumas
Deep MLP	1375,234	500,135	Geras MAE, tačiau gilesnė struktūra didina parametrų skaičių
Hybrid CNN-LSTM	1379,433	610,063	Vidutinis tikslumas, sudėtinga realizacija

Modelis (angl.)	RMSE, kW	MAE, kW	Vertinimas
RNN	1391,128	427,634	Geras MAE, bet reikalinga sekos būsena
ARIMA	1802,878	1498,732	Tinkamas kaip statistinis etalonas, bet silpnesnis tikslumas
MLP	1853,795	1523,615	Paprasčiausias neuroninis modelis, bet šioje iteracijoje ne pats tiksliausias
CNN	2689,214	1488,512	Prasčiausias RMSE tarp lygintų modelių

Iš rezultatų matyti, kad geriausias testavimo tikslumas Python aplinkoje buvo pasiektas naudojant RF modelį. Jo RMSE siekė 523,269 kW, o MAE – 124,241 kW. Tai reikšmingai geresnis rezultatas nei bazinio MLP modelio, kurio RMSE buvo 1853,795 kW, o MAE – 1523,615 kW. Taip pat gerą rezultatą pasiekė LSTM modelis, kurio RMSE buvo 864,279 kW, o MAE – 368,499 kW.

Tačiau šie rezultatai negali būti vertinami vien tik pagal paklaidą. Modelio tikslumas Python aplinkoje yra tik vienas kriterijus. Galutiniam taikymui svarbu, ar modelis gali būti perduotas į PLC ir ar jo skaičiavimo logika gali būti patikimai realizuota CODESYS ST kalba. Dėl to tiksliausias modelis nebūtinai yra geriausias praktiniam diegimui.

RF modelis pasiekė geriausią tikslumą, tačiau jo realizavimas PLC aplinkoje yra sudėtingesnis dėl modelio parametrų dydžio. Kiekvienas medis sudarytas iš mazgų, kuriuose reikia saugoti požymio numerį, slenkstinę reikšmę, kairiojo ir dešiniojo mazgo indeksus bei lapo reikšmę. Jei medžių daug arba jie gilūs, parametrų skaičius greitai išauga. Tuo tarpu MLP modelis turi aiškų ir kompaktišką parametrų rinkinį: sluoksnių svorius ir poslinkius. Jų skaičius tiesiogiai priklauso nuo įėjimų ir neuronų skaičiaus, todėl tokį modelį lengviau supakuoti, perduoti ir vykdyti PLC cikle.

LSTM, RNN, CNN ir hibridiniai modeliai taip pat parodė potencialą, tačiau jų praktinis taikymas PLC aplinkoje būtų dar sudėtingesnis. LSTM modelis turi vartų mechanizmus, vidines būsenas ir atminties vektorius, todėl jo realizacijai PLC reikėtų kur kas daugiau kodo ir tarpinių kintamųjų. CNN modelis reikalauja konvoliucijų, o hibridiniai CNN-LSTM arba RNN-MLP modeliai sujungia kelias sudėtingas skaičiavimo logikas. Dėl to šie modeliai šiame etape buvo laikomi vertingais tyrimo palyginimui, bet ne pagrindiniais kandidatais PLC realizacijai.

3.4.4. Modelių grafinis palyginimas

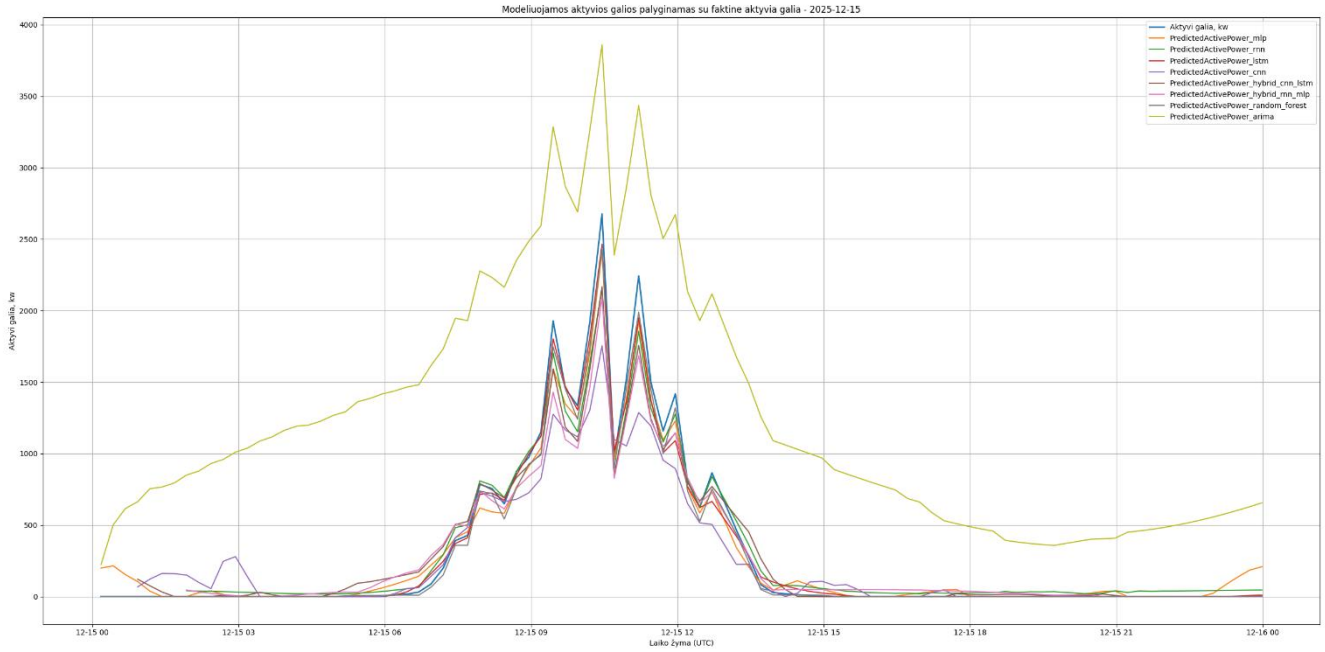
Papildomai prie bendro testavimo duomenų rinkinio buvo atliktas atskiras modelių palyginimas konkrečiai šaltojo laikotarpio parai. Tokia analizė reikalinga todėl, kad bendrosios metrikos parodo modelio vidutinį veikimą visoje testavimo imtyje, tačiau ne visada leidžia įvertinti, kaip modelis elgiasi konkrečiomis eksploatacinėmis sąlygomis. Šiam tikslui buvo pasirinkta 2025-12-15 diena. Kiekvienam modeliui buvo suformuota tos pačios paros prognozė, kuri vėliau palyginta su realia aktyviąja galia. Taip buvo galima įvertinti ne tik skaitines paklaidas, bet ir grafinį prognozės kreivės sutapimą su realia generacija.

10 lentelė. Matematinis modelių palyginimas 2025-12-15

Modelis (angl.)	MSE	RMSE, kW	MAE, kW	Vertinimas
RF	3617,597	60,146	27,685	Tiksliausias šios dienos modelis
LSTM	4801,850	69,295	30,710	Labai artimas RF rezultatui

Modelis (angl.)	MSE	RMSE, kW	MAE, kW	Vertinimas
MLP	9466,198	97,294	61,112	Stabilus ir praktiškai tinkamas rezultatas
RNN	10043,643	100,218	61,220	Artimas MLP rezultatui
Hybrid CNN-LSTM	15614,153	124,957	67,194	Vidutinis tikslumas
Hybrid RNN-MLP	22176,490	148,918	81,925	Prastesnis už bazinį MLP
CNN	45674,503	213,716	104,060	Didelė paklaida
ARIMA	1066706,073	1032,815	956,186	Netinkamas šios dienos prognozei

Grafinis visų modelių palyginimas pateikiamas 18 pav. Iš grafiko galima įvertinti, kaip skirtingų modelių prognozės seka realios aktyviosios galios kreivę. Šis palyginimas papildo skaitines metrikas, nes kai kurie modeliai gali turėti panašų MAE, tačiau skirtingai atkurti kreivės dinamiką. Pavyzdžiui, modelis gali gerai atitikti vidutinį galios lygį, bet vėluoti reaguodamas į staigius pokyčius arba per daug išlyginti generacijos kreivę. Tokie skirtumai ypač svarbūs realaus laiko PPC taikymui, nes valdikliui svarbu ne tik vidutinė dienos paklaida, bet ir momentinis modelio patikimumas.



20 pav. Visų modelių prognozių palyginimas su realia aktyviąja galia 2025-12-15 dieną

Atskiri modelių grafikai pateikiami 4-ąjame priede **(žr. 4 priedas)**. Juose detaliau matyti, kaip konkretus modelis atkartoja realios galios kreivę ir kuriose vietose atsiranda didžiausi nuokrypiai. Šie grafikai buvo naudojami ne tik formaliam rezultatų pateikimui, bet ir praktinei modelių atrankai, nes vien tik paklaidų lentelė neparodo visų valdymui svarbių modelio elgsenos savybių.

Šaltojo laikotarpio paros analizė patvirtino bendrą ankstesnio palyginimo tendenciją: RF modelis gali pasiekti labai aukštą tikslumą Python testavimo aplinkoje, tačiau jo praktinis pritaikymas PLC valdiklyje yra sudėtingas dėl didelio parametrų kiekio ir sprendimų medžių interpretavimo. LSTM

taip pat pasiekė gerą tikslumą, tačiau jo realizacijai PLC reikėtų saugoti sekos būsenas ir įgyvendinti sudėtingesnę skaičiavimo logiką. MLP modelis nebuvo pats tiksliausias, tačiau jo rezultatas buvo pakankamai geras, o modelio struktūra išlieka paprasčiausia eksportavimui ir vykdymui CODESYS aplinkoje.

3.4.5. Eksportuojamų modelių atranka

Po modelių tikslumo palyginimo buvo suformuota atskira eksportuojamų modelių logika. Kadangi ne visi modeliai vienodai tinkami PLC realizacijai, buvo nuspręsta į PLC eksportavimo etapą įtraukti tik tuos modelius, kuriuos įmanoma pakankamai aiškiai ir kompaktiškai aprašyti parametrų masyvu. Tokiais modeliais pasirinkti MLP, RF ir ARIMA. Toks pasirinkimas pagrįstas praktiniu diegimo požiūriu:

– MLP modelis gali būti aprašomas svorių ir poslinkių masyvais;
– RF modelis gali būti aprašomas sprendimų medžių mazgų struktūromis;
– ARIMA modelis gali būti aprašomas autoregresiniais, slankiojo vidurkio ir požymių koeficientais.

Kiti modeliai, tokie kaip LSTM, CNN, RNN ir hibridiniai tinklai, nors ir gali pasiekti gerą tikslumą, nebuvo pasirinkti tiesioginiam PLC eksportui, dėl realizacijos kompleksiškumo, ciklinėje programinėje aplinkoje. Tai didintų klaidų tikimybę ir apsunkintų sistemos priežiūrą.

Modelių atrinkimas

```
best_overall = _best_model_name(metrics)

exportable = {
    k: v for k, v in metrics.items()
    if k in SUPPORTED_ST_MODELS
}

best_exportable = _best_model_name(exportable)

if best_overall != best_exportable:
    logger.info(
        "Best overall model is '%s', but selected '%s' for PLC export.",
        best_overall,
        best_exportable,
    )
```

21 pav. Python kodo ištrauka, kuri parodo kaip atrenkamas geriausias modelis iš tinkamų naudoti PLC

Ši kodo dalis įgyvendina praktinį darbo principą. Jei tiksliausias modelis nėra palaikomas PLC eksporto logikoje, sistema atrenka tiksliausią iš tų modelių, kuriuos galima realiai perduoti ir vykdyti valdiklyje.

3.4.6. Bendro parametrų masyvo sudarymas

Kad skirtingus modelius būtų galima perduoti į PLC vienodu būdu, buvo sukurta bendra modelio parametrų pakavimo struktūra. Ankstesnė CODESYS realizacija buvo pritaikyta konkrečiam MLP modeliui ir turėjo fiksuotą sluoksnių struktūrą. Toks sprendimas nebetiko, kai atsirado poreikis perduoti ne tik MLP, bet ir RF arba ARIMA modelius. Todėl buvo suformuotas vienas bendras *REAL* tipo parametrų masyvas, kuriame pirmiausia įrašoma bendra modelio informacija, o po konkretaus modelio parametrai. Tokia struktūra leidžia PLC pusėje naudoti vieną priėmimo mechanizmą, o modelio interpretavimo logiką pasirinkti pagal *model_type* reikšmę.

11 lentelė. Modelio kodavimo kintamieji

Duomens pavadinimas	**Paskirtis**
„model_type“	Modelio tipo kodas: 1 – MLP, 2 – RF, 3 – ARIMA.
„param_count“	Perduodamų *REAL* reikšmių skaičius
„v“	Vienmatis *REAL* reikšmių masyvas su modelio parametrais

Pirmoji masyvo dalis yra bendra visiems modeliams. Joje saugomi išėjimo normalizavimo koeficientai, įėjimų skaičius, įėjimų vidurkiai ir skalės. Tai leidžia PLC pusėje atlikti tą patį įėjimų normalizavimą ir išėjimo denormalizavimą, kuris buvo naudojamas Python aplinkoje. Po bendros dalies pridedami konkretaus modelio parametrai. Pavyzdžiui, MLP atveju į masyvą įrašomas sluoksnių skaičius, neuronų skaičiai ir visi sluoksnių svoriai bei poslinkiai. Tai reiškia, kad CODESYS pusėje modelis nebėra pririštas prie vienos konkrečios struktūros. Jei modelio parametrai telpa į numatytą masyvą, galima perduoti kitokio dydžio MLP struktūrą.

Pirmoji masyvo dalis

```
def build_common_header(meta):
    x_mean = [float(v) for v in meta["x_scaler_mean"]]
    x_scale = [float(v) for v in meta["x_scaler_scale"]]
    y_mean = float(meta["y_scaler_mean"])
    y_scale = float(meta["y_scaler_scale"])
    input_count = len(x_mean)

    return [
        1.0,
        y_mean,
        y_scale,
        float(input_count),
        0.0,
        *x_mean,
        *x_scale,
    ]
```

Antroji masyvo dalis (MLP pavyzdys)

```
def load_mlp_payload(model_dir, meta):
    fc1_w = np.load(model_dir / "fc1_weight.npy")
    fc1_b = np.load(model_dir / "fc1_bias.npy")
    fc2_w = np.load(model_dir / "fc2_weight.npy")
    fc2_b = np.load(model_dir / "fc2_bias.npy")
    fc3_w = np.load(model_dir / "fc3_weight.npy")
    fc3_b = np.load(model_dir / "fc3_bias.npy")

    payload = [
        3.0,
        float(fc1_w.shape[0]),
        float(fc2_w.shape[0]),
        float(fc3_w.shape[0]),
    ]

    for weights, bias in [(fc1_w, fc1_b), (fc2_w, fc2_b), (fc3_w, fc3_b)]:
        out_dim, in_dim = weights.shape
        for i in range(out_dim):
            for j in range(in_dim):
                payload.append(float(weights[i, j]))
            payload.append(float(bias[i]))

    return payload
```

Antroji masyvo dalis (ARIMA pavyzdys)

```
payload: List[float] = [
 float(p),
 float(d),
 float(q),
 float(const_term),
 float(len(exog_params)),
 *ar_params,
 *ma_params,
 *exog_params,
]
```

Antroji masyvo dalis (RF pavyzdys)

```
def load_rf_payload(
 model_dir: Path,
 meta: Dict[str, Any]
) -> List[float]:
 if "random_forest" not in meta["models"]:
 raise RuntimeError("random_forest model not found in meta.json")

 model_path = model_dir / "models" / "random_forest" / "model.pkl"

 with open(model_path, "rb") as f:
 model = pickle.load(f)

 payload: List[float] = [float(len(model.estimators_))]

 for estimator in model.estimators_:
 payload.extend(_tree_to_payload(estimator.tree_))

 return payload
```

22 pav. Python kodo aplinkoje realizuotas modelių parametrų struktūrizavimas

Tai buvo svarbus žingsnis nuo rankiniu būdu realizuoto MLP modelio prie bendresnės modelių perdavimo sistemos. CODESYS pusėje modelio taikymo logika buvo perrašyta taip, kad pagal *model_type* būtų pasirenkama atitinkama interpretavimo šaka:

- MLP;
- RF;
- ARIMA.

Ši modelio taikymo logika pateikta 2 priede **(žr. 2 priedas)**.

Tokiu būdu CODESYS programa tapo universalesnė: tas pats priėmimo ir saugojimo mechanizmas gali būti naudojamas keliems modeliams, o skiriasi tik taikymo funkcija. Tai ypač svarbu automatiniam modelių atnaujinimui, nes PLC nebereikia rankiniu būdu perrašyti kiekvieną kartą, kai Python aplinkoje apmokomas naujas modelis.

3.4.7. Modelio parametrų perdavimas per MQTT

Sukūrus bendrą parametrų masyvą, kitas žingsnis buvo automatinis modelio perdavimas iš mokymo aplinkos į realų PPC valdiklį. Tam buvo pasirinkta MQTT komunikacija per „EMQX“ brokerį. Tokia komunikacijos schema leidžia Python mokymo aplinkai publikuoti modelio parametrus į konkrečią MQTT temą *(angl. topic)*, o PLC valdikliui juos gauti kaip MQTT prenumeratoriui.

Žinutės publikavimas

```
•payload = {
• "topic": topic,
• "payload": json.dumps(message)
•}

•response = _session.post(
• EMQX_API_URL,
• auth=HTTPBasicAuth(EMQX_API_KEY, EMQX_API_SECRET),
• json=payload,
• headers={"Content-Type": "application/json"},
• timeout=TIMEOUT,
•)
```

23 pav. Žinutės HTTP API užklausa

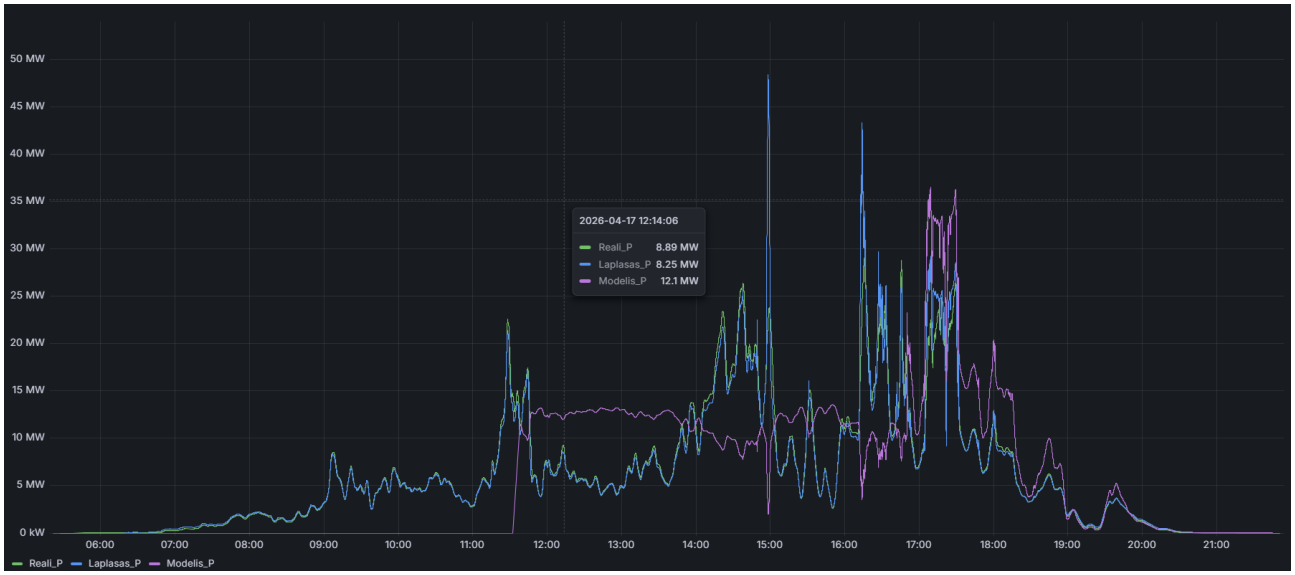
Šioje realizacijoje numatyta ir pakartotinių bandymų logika, jei įvyksta laikinas tinklo arba brokerio sutrikimas. Tai svarbu praktiniam naudojimui, nes modelio perdavimas į PLC neturi priklausyti nuo vienkartinės HTTP užklausos sėkmės.

CODESYS pusėje buvo parengta MQTT kliento logika, kuri priima publikaciją, išskiria modelio tipą, parametrų skaičių ir parametrų masyvą, o tada įrašo juos į valdiklio kintamuosius. Ši modelio gavimo per MQTT logika pateikta 3 priede **(žr. 3 priedas)**.

3.4.8. RF ir ARIMA modelių praktinio taikymo ribojimai

Nors RF modelis Python testavimo aplinkoje pasiekė geriausią tikslumą, jo taikymas PLC valdiklyje pasirodė problemiškas. Pagrindinė problema yra modelio dydis. Kad RF modelis būtų tikslus, jam reikia pakankamai daug medžių ir pakankamo jų gylio. Tačiau kiekvienas medis turi daug mazgų, o kiekvienam mazgui reikia kelių parametrų. Dėl to net vidutinio dydžio RF modelis greitai viršija parametrų kiekį, kurį patogu perduoti ir saugoti PLC valdiklyje.

Normaliai realizacijai Python aplinkoje naudotas RF modelis turėjo apie ~16000 parametrų, todėl tiesioginis visos tokio dydžio struktūros perkėlimas į PLC būtų nepraktiškas dėl valdiklio atminties. Dėl šios priežasties PLC realizacijai RF modelio dydis turėjo būti ribojamas. Mažesnis RF modelis išlaiko tą pačią sprendimų medžių veikimo logiką, todėl gali būti sutalpintas į PLC parametrų masyvą. Vis dėlto toks dydžio ribojimas turi tiesioginę įtaką tikslumui: kuo mažiau medžių ir kuo mažesnis jų gylis, tuo mažesnė modelio geba aprašyti sudėtingas nelinijines priklausomybes tarp apšvietos, temperatūros, laiko požymių ir aktyviosios galios. Todėl nors RF modelis Python aplinkoje pasiekė labai gerus rezultatus, jo praktinis taikymas PLC valdiklyje tapo ribotas dėl modelio dydžio ir atminties poreikio. Kas ir gali būti ryškiai matoma realizavus modelį realioje elektrinės aplinkoje su ribotu parametrų skaičiumi iki 1100 reikšmių.



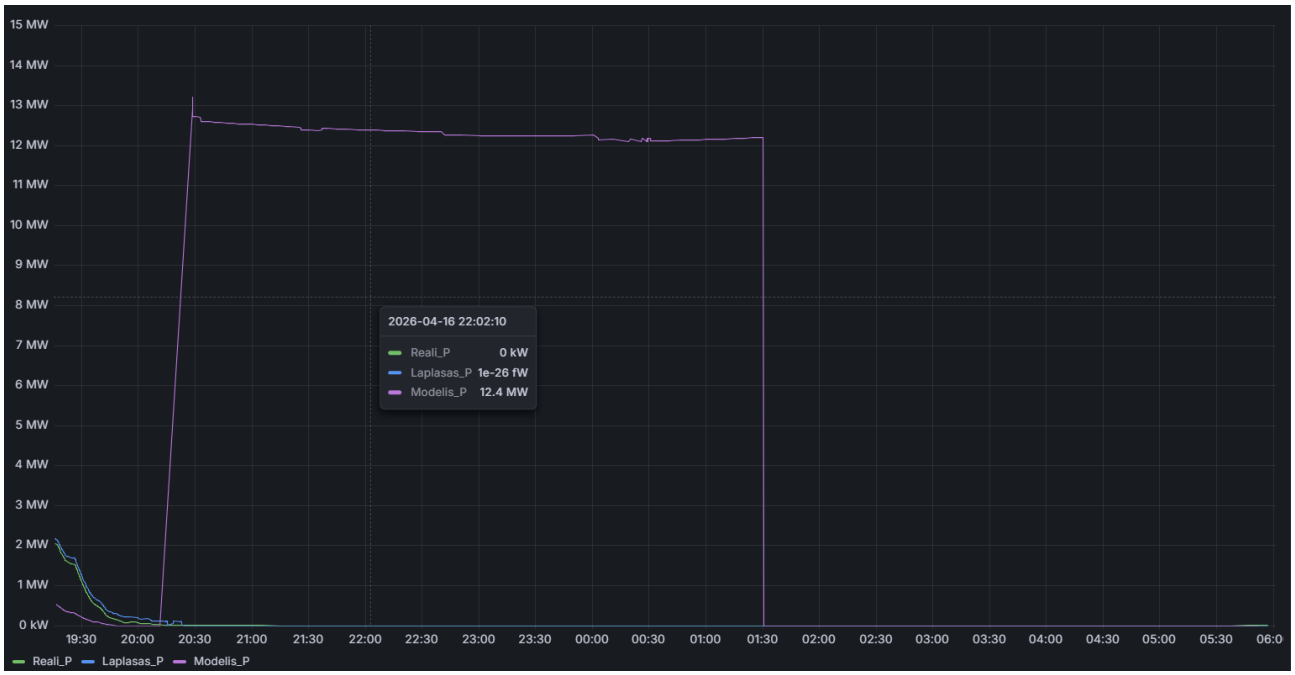
24 pav. RF modelio palyginimas su Laplaso modeliu ir tikra generacija realioje sistemoje

12 lentelė. Matematinis RF ir Laplaso modelių palyginimas su aktyvia galia realioje sistemoje

Modelis	**MAE, kW**	**RMSE, kW**	**Pastaba**
Laplasas	884,848	2275,715	Standartinis suderintas elektrinėje naudojamas modelis, šioje paroje pasirinktas kaip etalonas.
RF	5731,278	7142,911	PLC realizacijai modelio dydis ribotas iki 1100 reikšmių, todėl tikslumas reikšmingai prastesnis už pilną RF modelį Python aplinkoje.

ARIMA modelio taikymas taip pat turėjo praktinių apribojimų. ARIMA yra laiko eilučių modelis, todėl jo skaičiavimas priklauso ne tik nuo dabartinių įėjimų, bet ir nuo ankstesnių išėjimų bei ankstesnių klaidų. Tai reiškia, kad PLC pusėje reikia saugoti modelio istoriją, atnaujinti grįžtamojo ryšio reikšmes ir užtikrinti, kad modelis būtų vykdomas nuosekliai kiekviename cikle. Jei ciklas praleidžiamas, jei duomenys vėluoja arba jei valdiklyje nėra patikimo grįžtamojo ryšio, ARIMA rezultatas gali tapti nestabilus. Papildoma problema yra ta, kad ARIMA modelio korektiškam veikimui realioje elektrinėje reikalinga patikimai žinoti, kada elektrinė veikia neribotu režimu, o kada aktyvioji galia yra ribojama valdymo sistemos. Python mokymo aplinkoje ribojimo būsenos buvo aptinkamos ir filtruojamos pagal apšvietos ir galios gaubtinės metodą, tačiau PLC valdiklyje tokia pilna ribojimo aptikimo logika nėra tiesiogiai realizuota. Valdiklio aplinkoje ne visada pakanka vien užduoties reikšmės, nes elektrinė gali nedageneruoti dėl kitų priežasčių, galia gali svyruoti apie

užduotį dėl PID reguliatoriaus, taip pat gali veikti rampos ar kiti pereinamieji režimai. Dėl to ARIMA modelis PLC pusėje gali atnaujinti savo būseną pagal duomenis, kurie neatspindi realaus maksimalios generacijos potencialo. Ši problema ypač aktuali realiame PPC, nes modelis turi veikti kartu su elektrinės valdymo logika ir realaus laiko ribojimais. ARIMA modelis nėra vien paprasta formulė pagal apšvietą ir temperatūrą. Dėl to jo integracija į PLC yra sudėtingesnė, o testavimo ir derinimo procesas tampa mažiau patikimas.



25 pav. ARIMA modelio reakcija realioje elektrinėje

13 lentelė. Matematinis ARIMA ir Laplaso modelių palyginimas su aktyvia galia realioje sistemoje

Modelis	**MAE, kW**	**RMSE, kW**	**Pastaba**
Laplasas	33,118	57,707	Standartinis suderintas elektrinėje naudojamas modelis, šioje paroje pasirinktas kaip etalonas.
ARIMA	8019,198	9815,395	PLC aplinkoje modelis veikė nestabiliai: net naktį, kai reali galia artima 0 kW, prognozė išliko labai didelė.

3.4.9. MLP modelio pasirinkimas tolimesnei dockerizacijai

Apibendrinus daugelio modelių tyrimą, buvo padaryta išvada, kad tolimesniam dockerizavimo ir automatinio permokymo etapui racionaliausia naudoti MLP modelį. Nors šioje konkrečioje testavimo iteracijoje RF pasiekė geriausią paklaidą, jo praktinis taikymas PLC aplinkoje yra sudėtingas dėl didelio parametrų kiekio. ARIMA modelis taip pat buvo laikomas eksportuojamu kandidatu, tačiau jo priklausomybė nuo grįžtamojo ryšio ir ankstesnių reikšmių apsunkina stabilų taikymą realaus laiko valdiklyje. MLP modelis pasirinktas dėl šių priež

– jis turi paprastą ir aiškią struktūrą;
– jo parametrai lengvai eksportuojami kaip svoriai ir poslinkiai;
– PLC aplinkoje realizuojama keliomis matricų daugybos operacijomis;
– nereikia saugoti ilgos istorijos arba sudėtingos vidinės būsenos;
– modelio struktūrą galima keisti, jei nauji sluoksnių dydžiai telpa į parametrų masyvą;

– CODESYS pusėje jau buvo patikrinta MLP skaičiavimo logika;
– modelis geriausiai atitinka darbo tikslą – rasti ne tik tikslų, bet ir realiai eksploatacijai tinkamą sprendimą.

14 lentelė. Modelių diegimo palyginimas

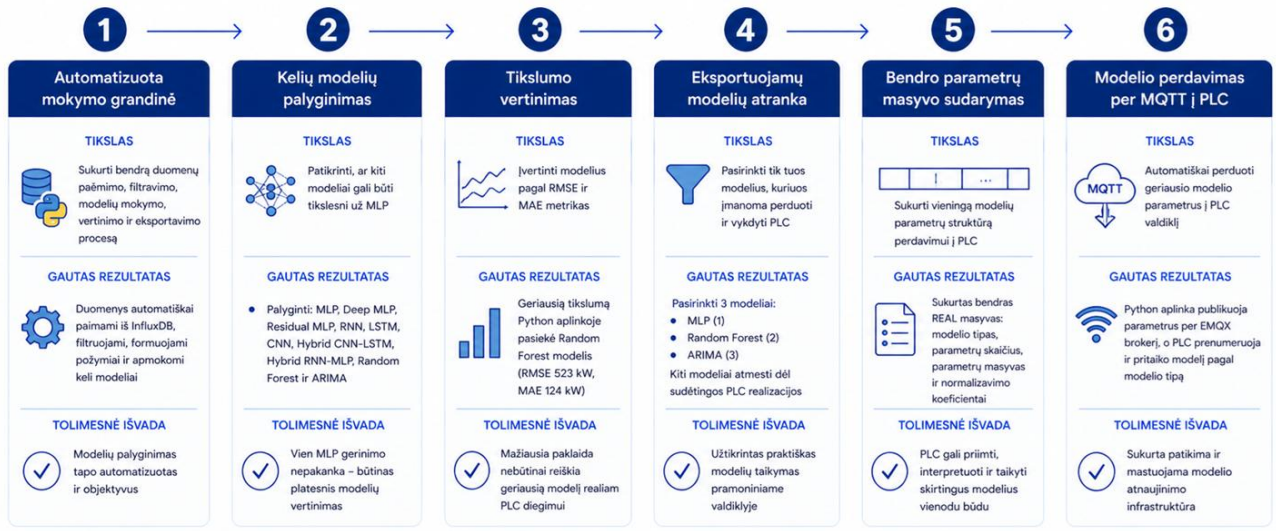
Modelis	**Tikslumas**	**PLC realizavimo sudėtingumas**	**Parametrų perdavimas**	**Būsenų / sekos poreikis**	**Galutinis vaidmuo**
MLP	geras	mažas	paprastas	nėra	diegiamas
RF	gali būti geras	vidutinis / didelis	daug medžių parametrų	nėra	testavimui / ribotas
ARIMA	nestabilus realiai sistemai	vidutinis	sudėtingesnė būsena	reikia ankstesnių reikšmių	palyginimui
LSTM	galimai geras	didelis	daug parametrų	reikia sekos/būsenos	teorinis
CNN	eksperimentinis	didelis	struktūriškai sudėtingesnis	reikia specialiai formuoti įėjimą	teorinis
hibridinis	eksperimentinis	didelis	struktūriškai sudėtingesnis	reikia sekos / kelių modelių išvesčių	teorinis

Svarbu pažymėti, kad RF ir ARIMA modeliai nebuvo visiškai atmesti. Python ir CODESYS struktūroje palikta galimybė juos kurti, eksportuoti ir priimti per MQTT. Tai reiškia, kad ateityje, jei būtų padidintas leidžiamas parametrų kiekis arba dalis skaičiavimo būtų perkelta į „edge“ įrenginį, šiuos modelius būtų galima vėl įtraukti į praktinius bandymus. Vis dėlto šiame darbo etape, ruošiantis modelio mokymo logiką perkelti į Linux VM ir dockerizuoti, pagrindiniu modeliu pasirinktas MLP. Taip išlaikomas sistemos paprastumas, bet kartu paliekama galimybė ateityje naudoti ir kitus modelius.

3.4.10. Trečiojo tyrimo etapo apibendrinimas

Šiame tyrimo etape buvo pereita nuo vieno stabilizuoto MLP modelio prie platesnio modelių palyginimo. Buvo sukurta automatizuota mokymo grandinė, kuri paima istorinius duomenis iš „InfluxDB“, pritaiko ribojimo filtrą, suformuoja požymius, apmoko kelias modelių šeimas, apskaičiuoja jų paklaidas ir eksportuoja modelių artefaktus. Palyginus modelius nustatyta, kad Python aplinkoje geriausią tikslumą pasiekė RF modelis, tačiau jo praktinis taikymas PLC aplinkoje yra ribotas dėl didelio parametrų skaičiaus. Sekų modeliai, tokie kaip LSTM, RNN ir hibridiniai tinklai, parodė gerą prognozavimo potencialą, tačiau jų realizacija PLC aplinkoje būtų sudėtinga dėl vidinių būsenų, sekų saugojimo ir didesnio skaičiavimo kiekio. ARIMA modelis buvo įvertintas kaip galimas statistinis kandidatas, tačiau jo priklausomybė nuo ankstesnių reikšmių ir grįžtamojo ryšio apsunkina stabilų taikymą realiuoju laiku.

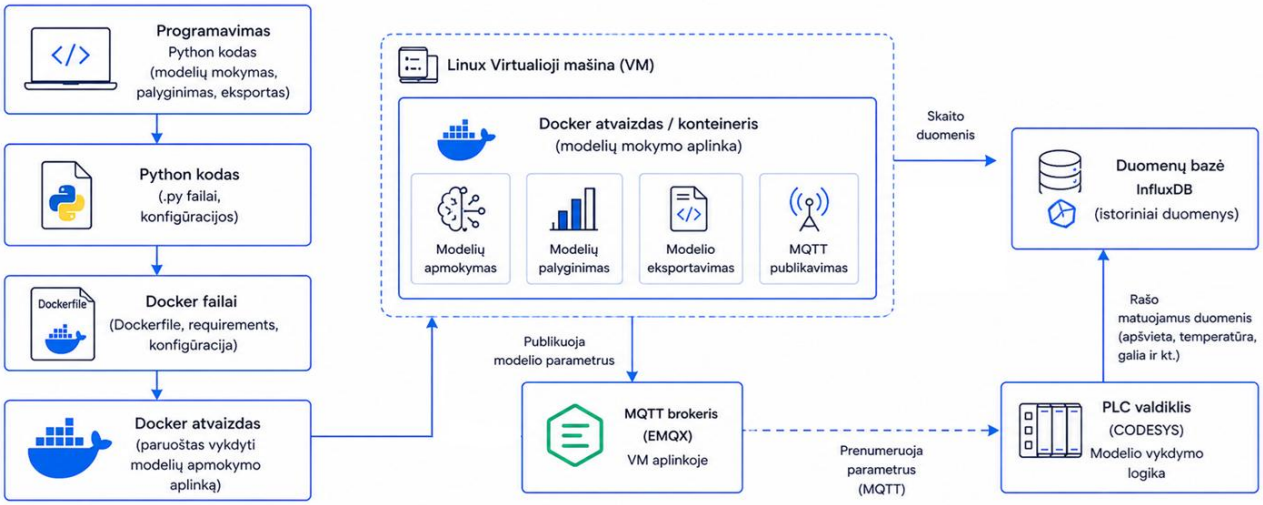
Dėl šių priežasčių tolimesniam darbo etapui pasirinktas MLP modelis. Nors jis nebuvo tiksliausias visų modelių palyginime, jis geriausiai atitiko bendrą darbo tikslą: sukurti tokį prognozavimo modelį, kuris būtų pakankamai tikslus, lengvai eksportuojamas, patikimai perduodamas per MQTT ir realiai vykdomas PLC valdiklyje.



26 pav. Trečiojo tyrimo etapo eiga

3.5. Galutinės automatinės modelio mokymo infrastruktūros dockerizacija ir diegimas Linux VM aplinkoje

Ankstesniame tyrimo etape buvo parodyta, kad tiksliausias modelis Python aplinkoje nebūtinai yra tinkamiausias PLC realizacijai. RF ir LSTM modeliai kai kuriose testavimo imtyse pasiekė gerą tikslumą, tačiau jų realizacija PLC aplinkoje reikalautų didesnio parametrų kiekio, būsenų saugojimo arba sudėtingesnės interpretavimo logikos. Todėl paskutiniame tyrimo etape buvo pereita nuo plataus modelių palyginimo prie vieno praktiškai diegiamo modelio, MLP, automatinės mokymo ir publikavimo infrastruktūros kūrimo. Tokiu būdu sistema turėjo tapti ne vien tyrimo įrankiu, bet ir praktiniu prototipu, kurį galima paleisti Linux VM aplinkoje kaip nuolat veikiantį servisą.



27 pav. Galutinė modelių kūrimo struktūra

3.5.1. „PyTorch“ bibliotekos atsisakymas ir konteinerio dydžio optimizavimas

Pirmoji problema, iškilusi bandant sprendimą perkelti į „Docker“ aplinką, buvo konteinerio dydis. Ankstesnėse programos versijose MLP ir kiti neuroniniai tinklai buvo kuriami naudojant „PyTorch“ biblioteką. Lokaliems bandymams ši biblioteka buvo patogi, nes leido greitai kurti skirtingas neuroninių tinklų struktūras, naudoti GPU / CPU skaičiavimus ir aiškiai valdyti mokymo procesą.

Tačiau dockerizuojant sprendimą išryškėjo praktinis trūkumas: ši biblioteka kartu su priklausomybėmis labai padidino „Docker“ atvaizdo dydį. Pirminis konteineris užėmė apie 20 GB, jo kūrimas truko ilgai ir toks sprendimas nebuvo patogus diegti į VM aplinką.

Kadangi galutiniame sprendime buvo nuspręsta naudoti tik MLP modelį, „PyTorch“ nebebuvo būtinas. MLP regresorių galima realizuoti naudojant lengvesnę *scikit-learn* biblioteką. Dėl to galutinėje programos versijoje MLP modelis realizuotas per *sklearn.neural_network.MLPRegressor*, o ne per „PyTorch“. Tai leido išlaikyti tą pačią pagrindinę modelio logiką – daugiasluoksnį perceptroną su ReLU aktyvacija – bet reikšmingai sumažinti atvaizdo dydį ir supaprastinti priklausomybes.

Galutinėje *requirements.txt* versijoje paliktos tik reikalingos bibliotekos:
– *numpy;*
– *pandas;*
– *matplotlib;*
– *scikit-learn;*
– *statsmodels;*
– *influxdb3-python;*
– *requests;*
– *apscheduler.*
Tai rodo, kad galutinė sistema buvo optimizuota būtent ilgalaikiam veikimui VM aplinkoje, o ne plačiam eksperimentiniam visų neuroninių architektūrų testavimui.

Galutinė „Docker“ failo struktūra taip pat tapo paprastesnė. Atvaizdas kuriamas lengvu „python:3.12-slim“ pagrindu. Į konteinerį nukopijuojami *requirements.txt* ir *app*, o paleidimo komanda tiesiog vykdo „scheduled_pipeline.py“ failą.

Konteinerizavimo instrukcijos

```
•FROM python:3.12-slim

•WORKDIR /app

•COPY requirements.txt .
•RUN pip install --no-cache-dir -r requirements.txt

•COPY app /app

•CMD ["python", "scheduled_pipeline.py"]
```

28 pav. *dockerfile* konteinerizavivmo instrukcijos „Docker“ atvaizdo sukūrimui

Pakeitus biblioteką ir supaprastinus „Docker“ atvaizdą, galutinė atvaizdo versija užėmė apie 1,64 GB. Tai yra esminis praktinis pagerinimas, nes toks atvaizdas jau gali būti patogiai saugomas, perkeliamas ir paleidžiamas Linux VM aplinkoje. Be to, sumažėja atnaujinimo laikas ir resursų naudojimas, o tai svarbu, kai sistema turi veikti kaip ilgalaikis automatinis servisas.

3.5.2. „Docker“ ir Linux VM infrastruktūros paruošimas

„Docker“ šiame darbe naudojamas tam, kad visa modelio mokymo programa veiktų izoliuotoje, lengvai perkeliamoje aplinkoje. Atvaizdas aprašo, kokia operacinė aplinka, Python versija, bibliotekos ir programos failai reikalingi sistemai paleisti. Tokio sprendimo privalumas yra tai, kad programa tampa nepriklausoma nuo konkrečios VM konfigūracijos. Jei Linux serveryje yra įdiegtas

„Docker“, tą patį atvaizdą galima paleisti nepriklausomai nuo to, kokios bibliotekos yra įdiegtos pačiame serveryje. Tai sumažina diegimo klaidų tikimybę ir leidžia lengviau atnaujinti programą.

Šiame darbe „Docker“ atvaizdas buvo diegiamas Linux VM aplinkoje. VM naudojama kaip atskiras skaičiavimo mazgas, kuriame vykdomas modelio apmokymas, modelių palyginimas, eksportavimas ir MQTT publikavimas. Tokia architektūra leidžia nenaudoti PLC resursų modelio mokymui. PLC valdiklyje lieka tik modelio vykdymo logika, o visi sunkūs skaičiavimo darbai atliekami VM aplinkoje.

29 pav. Pagrindinių „Docker“ komandų aprašymas

Būtent tokia logika buvo naudojama tikrinant realizuotą sprendimą VM aplinkoje. Išrašuose matyti, kad konteineris paleidžiamas kaip suplanuota užduotis, inicijuojamas *scheduled_pipeline.py*, užklausiami istoriniai duomenys, vykdomas modelio mokymas, lyginamas naujas modelis su anksčiau publikuotu ir, jei modelis pagerėja, jis publikuojamas per MQTT.

3.5.3. Konfigūravimas per ENV parametrus

Vienas svarbiausių galutinės versijos pakeitimų buvo visų pagrindinių parametrų perkėlimas į aplinkos kintamuosius. Tai reiškia, kad norint pakeisti duomenų bazės adresą, modelio mokymo parametrus, ribojimo filtro slenksčius, MQTT nustatymus ar modelio publikavimo dažnį, nereikia perrašyti programos kodo ir iš naujo kurti „Docker“ atvaizdo.

Parametrų nuskaitymas iš ENV

```
INFLUX_BUCKET = os.getenv("INFLUX_BUCKET")

USE_ENVELOPE_CURTAILMENT_FILTER = _get_bool_env(
 "USE_ENVELOPE_CURTAILMENT_FILTER",
 True
)

ENVELOPE_MIN_IRRADIANCE = float(
 os.getenv("ENVELOPE_MIN_IRRADIANCE", "50.0")
)

MAX_EXPORT_REALS = int(
 os.getenv("MAX_EXPORT_REALS", "1100")
)

TRAIN_RF = _get_bool_env("TRAIN_RF", False)
TRAIN_ARIMA = _get_bool_env("TRAIN_ARIMA", False)

MQTT_TOPIC_PREFIX = os.getenv("MQTT_TOPIC_PREFIX", "plants/NN_params")
MQTT_TOPIC_SUFFIX = os.getenv("MQTT_TOPIC_SUFFIX", "Test_NN")
```

ENV failo ištrauka

```
USE_ENVELOPE_CURTAILMENT_FILTER=true
ENVELOPE_MIN_IRRADIANCE=0
ENVELOPE_QUANTILE=0.98
CURTAILMENT_RATIO_THRESHOLD=0.95

BATCH_SIZE=512
LEARNING_RATE=0.0002
EPOCHS=400
MAX_EXPORT_REALS=1100

TRAIN_RF=false
TRAIN_ARIMA=false

SCHEDULER_TIMEZONE=UTC
SCHEDULE_HOUR_UTC=0
SCHEDULE_MINUTE_UTC=0
RUN_ON_STARTUP=true

MQTT_TOPIC_PREFIX=plants/NN_params
MQTT_TOPIC_SUFFIX=Test_NN
```

30 pav. Kodo ištrauka. Parametrų nuskaitymas iš ENV

Toks sprendimas yra svarbus eksploataciniu požiūriu. Pavyzdžiui, jei pastebima, kad ribojimo filtras pašalina per daug eilučių, galima pakeisti „ENVELOPE_MIN_IRRADIANCE“ arba „ENVELOPE_QUANTILE“ reikšmes nekeičiant programos. Jei reikia sumažinti VM apkrovą, galima sumažinti architektūrų paieškos apimtį arba pakeisti mokymo parametrus. Jei modelis turi būti publikuojamas į kitą elektrinę, pakanka pakeisti MQTT temą *(angl. topic)*.

Tokia struktūra leido modelio mokymo sistemą eksploatuoti kaip konfigūruojamą servisą: keičiant ENV vertes galima reguliuoti mokymo laiką, „InfluxDB“ šaltinį, MQTT temą, ribojimo filtro parametrus ir modelio architektūrų paieškos ribas nekeičiant programos kodo.

3.5.4. Duomenų praradimo problemos sprendimas interpoliuojant signalus

Ankstesnėje tyrimo dalyje **(žr. 3.3.5)** buvo pastebėta, kad modelio mokymo imtis labai sumažėja dėl duomenų eilučių apjungimo problemos. Aktyvioji galia, apšvieta ir temperatūra ne visada į duomenų bazę įrašomos tiksliai tuo pačiu laiko momentu. Jei tokie signalai jungiami tik pagal identišką laiko žymą, o po to taikoma *dropna*, pašalinama labai didelė dalis duomenų. Tai ypač blogino modelio su temperatūros požymiu mokymą, nes temperatūros signalas turėjo savo atskirą įrašinėjimo dažnį.

Galutinėje programos versijoje ši problema buvo sprendžiama pakeičiant duomenų užklausimo ir sulyginimo logiką. Duomenys iš „InfluxDB“ pirmiausia agreguojami 1 sekundės intervalu naudojant *date_bin*, o po to Python pusėje sudaroma tanki 1 sekundės laiko ašis. Aktyviosios galios, apšvietos ir temperatūros signalai perindeksuojami į šią bendrą laiko ašį, o trūkstamos reikšmės interpoliuojamos **(žr. 29 pav.)**.

Duomenų interpoliacija

```
dense_index = pd.date_range(
 start=df["timestamp"].min(),
 end=df["timestamp"].max(),
 freq="1s",
 tz="UTC",
)

df = df.set_index("timestamp").reindex(dense_index)
df.index.name = "timestamp"

df[["ActivePower", "Irradiance", "Temperature"]] = (
 df[["ActivePower", "Irradiance", "Temperature"]]
 .interpolate(method="time", limit_direction="both")
)

df = df.reset_index()
```

31 pav. Kodo ištrauka. Duomenų sulyginimas ir interpoliacija

Interpoliavimas atliktas tiesiniu metodu pagal laiko žymą, suvienodinant signalus į bendrą laiko gardelę. Trūkstamos reikšmės buvo užpildomos tik tada, kai tarpai buvo pakankamai trumpi; ilgesni duomenų praradimai neturėtų būti interpoliuojami, nes tai kurtų dirbtinius mokymo duomenis.

Tai leido išsaugoti žymiai daugiau mokymo duomenų ir kartu naudoti temperatūros požymį neprarandant didžiosios dalies imties. Bandymo su pakeista duomenų sulyginimo logika metu galutinis sujungtas duomenų rinkinys turėjo apie 25 milijonus eilučių, o chronologinis padalinimas sudarė 11788710 mokymo, 1309856 validavimo ir 1455396 testavimo eilučių. Palyginimui, ankstesniame bandyme su platesne MLP paieška, iš 25 milijonų eilučių po filtravimo mokymui liko tik 272912 eilutės, validavimui 30323, o testavimui 33692 eilutės. Taigi kai signalai buvo sulyginti ir interpoliuoti, modelis gavo daug reprezentatyvesnę mokymo imtį, apimančią daugiau skirtingų elektrinės darbo sąlygų.

Realių telemetrijos duomenų paruošimo etape buvo įgyvendintas signalų sinchronizavimas ir trūkstamų reikšmių interpoliavimas. Tai nėra atskiras prognozavimo metodas, tačiau tai būtina praktinė duomenų paruošimo sąlyga, leidžianti formuoti vientisą mokymo imtį ir užtikrinti, kad modelio įėjimai bei tikslinė reikšmė būtų lyginami tame pačiame laiko momente.

3.5.5. MLP architektūros paieškos išplėtimas

Galutinėje MLP mokymo logikoje buvo išplėsta architektūrų paieška. Vietoje fiksuotos dviejų sluoksnių struktūros pradėta tikrinti platesnė MLP šeima. Programoje formuojami kandidatai su dviem, trimis arba keturiais paslėptais sluoksniais. Pirmųjų sluoksnių neuronų skaičius ribojamas nedidelėmis reikšmėmis, kad modelis išliktų tinkamas PLC eksportui.

Kodo logikoje MLP architektūrų kandidatai sudaromi taip: pirmasis sluoksnis tikrinamas intervale 6–8 neuronai, antrasis 3–5 neuronai, o trečias ir ketvirtas sluoksniai gali būti nuo 0 iki 3 neuronų. Jei trečio sluoksnio nėra, ketvirtas taip pat nepridedamas. Tokiu būdu gaunama plati, bet vis dar kompaktiška MLP architektūrų paieška

MLP architektūrų kandidatų sudarymas

```
for l1 in range(6, 9):
    for l2 in range(3, 6):
        for l3 in range(0, 4):
            for l4 in range(0, 4):
                layers = [l1, l2]

            if l3 > 0:
                layers.append(l3)

            if l4 > 0 and l3 > 0:
                layers.append(l4)

            layer_candidates.append(tuple(layers))
```

Per didelių arhitektūrų atmetimas

```
def estimate_mlp_packed_param_count(input_dim, hidden_layers):
    layer_sizes = [input_dim, *hidden_layers, 1]
    network_payload_header = 1 + len(layer_sizes)

    network_payload_weights = sum(
        (layer_sizes[i] * layer_sizes[i + 1]) + layer_sizes[i + 1]
        for i in range(len(layer_sizes) - 1)
    )

    common_header = 5 + 2 * input_dim
    return common_header + network_payload_header + network_payload_weights
```

32 pav. MLP architektūrų kandidatų sudarymas

Svarbu paminėti, kad kartu kiekvienai architektūrai apskaičiuojamas prognozuojamas eksportuojamų parametrų skaičius. Jei modelis viršija „MAX_EXPORT_REALS“ ribą, jis nėra treniruojamas. Tokiu būdu architektūros paieška iš karto susiejama su PLC realizavimo reikalavimu.

3.5.6. Duomenų kiekio ir mokymo parametrų įtakos MLP modelio tikslumui vertinimas

Išsprendus pirminę duomenų sulyginimo problemą, buvo atliktas papildomas tarpinis bandymas, kurio tikslas buvo atskirti du skirtingus veiksnius: MLP architektūros paieškos platumą ir mokymo duomenų kiekio įtaką galutiniam modelio tikslumui. Ankstesnėje iteracijoje buvo vykdyta plati MLP architektūrų paieška, kurios metu patikrinta 117 skirtingų neuroninio tinklo struktūrų. Vis dėlto tuo metu duomenų apjungimo problema dar nebuvo pilnai išspręsta, todėl po filtravimo mokymui liko santykinai nedidelė duomenų dalis. Šio bandymo metu modelių mokymui buvo naudojamos 272 912 eilutės. Nors architektūrų paieška buvo plati, galutinis modelis pasiekė RMSE = 1147,486 kW ir MAE = 716,135 kW.

Tolimesnėje iteracijoje buvo nuspręsta laikinai sumažinti architektūrų paieškos apimtį ir patikrinti, kokią įtaką modeliui daro ne struktūrų skaičius, o ženkliai padidėjęs mokymo duomenų kiekis. Tam buvo pasirinkta viena bazinė MLP architektūra, tačiau ji buvo mokoma su skirtingais mokymo parametrų deriniais. Iš viso buvo patikrinti 6 mokymo variantai, keičiant reguliarizacijos koeficientą alpha ir mokymosi greitį. Tokiu būdu buvo galima įvertinti ne tik pačios architektūros, bet ir mokymo proceso parametrų įtaką rezultatui.

Šiame bandyme duomenys jau buvo sulyginti bendroje 1 sekundės laiko ašyje ir interpoliuoti, todėl mokymo imtis tapo ženkliai didesnė. Modeliui mokyti naudota 11 788 710 eilučių. Nors šiuo atveju nebuvo vykdoma plati 117 architektūrų paieška, o tik vienos struktūros mokymas su 6 skirtingais mokymo parametrų rinkiniais, gautas rezultatas buvo geresnis: RMSE sumažėjo iki 1036,053 kW, o MAE – iki 514,958 kW.

Šis palyginimas parodė, kad šiame tyrimo etape modelio tikslumui didesnę įtaką turėjo ne vien architektūrų paieškos platumas, o mokymo duomenų kiekis ir jų reprezentatyvumas. Plati architektūrų paieška negalėjo kompensuoti per mažos mokymo imties. Tuo tarpu viena MLP

struktūra, apmokyta su daug didesniu duomenų kiekiu, pasiekė geresnį rezultatą net ir be plačios architektūrų paieškos. Taip pat buvo patvirtinta mokymo parametrų reikšmė. Per mažas mokymosi greitis gali sulėtinti mokymą, o per didelis – lemti nestabilią konvergenciją arba prastesnį minimumą. Dėl to buvo nuspręsta, kad finalinėje sistemos versijoje neužtenka tik ieškoti skirtingų MLP sluoksnių struktūrų – būtina kartu vertinti ir mokymo parametrų kombinacijas.

3.5.7. Tik geresnio modelio publikavimo logika

Kadangi sistema turi veikti automatiškai, būtina užtikrinti, kad kiekvieno naujo mokymo ciklo rezultatas nebūtų aklai perduodamas į PLC. Jei naujai apmokytas modelis prastesnis už jau naudojamą, jo publikavimas pablogintų realios sistemos veikimą. Dėl to galutinėje versijoje buvo pridėta publikuoto modelio būsenos saugojimo ir palyginimo logika. Programa saugo informaciją apie jau publikuotą modelį:

– modelio tipą;
– RMSE;
– MAE;
– MQTT temą;
– parametrų skaičių.

Kiekvieno naujo mokymo ciklo metu naujas geriausias eksportuojamas modelis lyginamas su anksčiau publikuotu. Jei naujas RMSE nėra mažesnis, MQTT publikavimas praleidžiamas. Jei naujas modelis geresnis, jis pakeičia ankstesnį publikuotą modelį ir jo parametrai išsiunčiami į PLC.

Modelio palyginimas

```
previous_state = _load_published_state(model_dir)

best_new_rmse = exportable[best_exportable]["test_rmse_kw"]
previous_rmse = (
    float(previous_state["test_rmse_kw"])
    if previous_state
    else float("inf")
)

if previous_state is not None and best_new_rmse >= previous_rmse:
 logger.info(
    "Best new model '%s' RMSE=%.3f kW does not beat "
    "published RMSE=%.3f kW. MQTT publish skipped.",
    best_exportable,
    best_new_rmse,
    previous_rmse,
 )
 return
```

33 pav. Naujo modelio palyginimas su publikuotu modeliu

Ši logika buvo patikrinta praktiniuose bandymuose. Viename iš bandymų naujas modelis pasiekė RMSE = 1147,486 kW, tačiau anksčiau publikuotas modelis turėjo RMSE = 1036,053 kW. Sistema teisingai nusprendė, kad naujas modelis nėra geresnis, todėl MQTT publikavimas buvo praleistas.

3.5.8. Galutinis dockerizuotos sistemos bandymas VM aplinkoje

Finalinis VM bandymas buvo atliktas sujungiant ankstesnių dviejų iteracijų išvadas: naudotas interpoliuotas didelės apimties duomenų rinkinys ir vėl įjungta plati 117 MLP architektūrų paieška. Galutinėje iteracijoje sistema buvo paleista jau pačioje Linux VM aplinkoje kaip „Docker“ konteineris. Šio bandymo tikslas buvo patikrinti visą grandinę realiomis sąlygomis: duomenų užklausimą iš „InfluxDB“, interpoliuotą duomenų paruošimą, ribojimo filtravimą, plačią MLP architektūrų paiešką, modelio palyginimą su anksčiau publikuotu modeliu ir MQTT publikavimą.

Konteinerio išrašuose matyti, kad konteineris buvo paleistas 2026-04-20 10:13:21 UTC, o modelio publikavimas baigtas 2026-04-22 04:00:14 UTC. Tai reiškia, kad visas procesas, įskaitant duomenų surinkimą, didelės imties paruošimą, plačią 117 MLP architektūrų paiešką ir modelio publikavimą, truko apie 41 val. 47 min. Toks laikas yra žymiai ilgesnis nei ankstesnių lokalių bandymų, tačiau jis paaiškinamas tuo, kad modelis buvo mokomas su labai dideliu duomenų kiekiu ir plačia architektūrų paieška.

15 lentelė. Galutinių dockerizuotos MLP sistemos iteracijų palyginimas

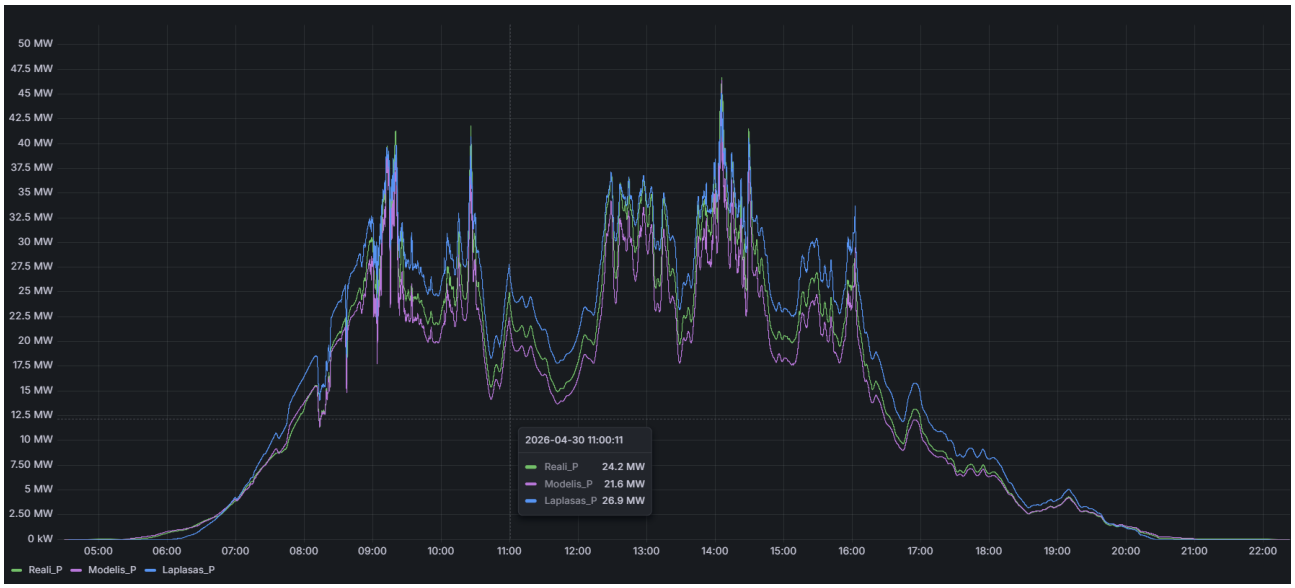
Iteracija	**Duomenų paruošimas**	**Paieškos apimtis**	**Mok. / Val. / Test.**	**RMSE, kW**	**MAE, kW**	**Publikavimo rezultatas**
Dar neišspręstas eilučių praradimas	Dalis eilučių prarandama dėl nesutampančių laiko žymų	117 MLP architektūrų	272912 / 30323 / 33692	1147,486	716,135	Nepublikuota, nes blogiau už ankstesnį modelį
Interpoliacija, viena MLP struktūra	Signalai sulyginami 1 s laiko ašyje	6 variantai	11788710 / 1309856 / 1455396	1036,053	514,958	Publikuota kaip pirmas geresnis modelis
Galutinis bandymas	Interpoliacija + plati MLP paieška	117 MLP architektūrų	11771565 / 1307951 / 1453279	540,331	157,201	Publikuota, nes pagerino ankstesnį modelį

34 pav. Galutinės sistemos sukurto modelio realizacija veikiančioje elektrinėje palyginti su Laplaso transformacijos prognoze ir tikra aktyvia galia

16 lentelė. 2026-4-19 MLP ir Laplaso modelio matematinis palyginimas su realia aktyvia galia

Modelis	**MAE, kW**	**RMSE, kW**	**Pastaba**
Laplasas	2778,852	3390,015	Standartinis suderintas elektrinėje naudojamas modelis, šioje paroje pasirinktas kaip etalonas.

Modelis	MAE, kW	RMSE, kW	Pastaba
MLP	1625,538	2190,893	Galutinės dockerizuotos sistemos sugeneruotas MLP modelis veikė stabiliai ir tiksliau atkartojo realią aktyviąją galią nei Laplaso modelis. Mažesnės MAE ir RMSE reikšmės rodo, kad modelio mokymo, eksportavimo, perdavimo į PLC ir taikymo grandinė buvo sėkmingai realizuota realioje sistemoje.



35 pav. Galutinės sistemos sukurto naujesnio modelio realizacija veikiančioje elektrinėje palyginti su Laplaso transformacijos prognoze ir tikra aktyvia galia

36 pav. Galutinės sistemos sukurto naujesnio modelio realizacija veikiančioje elektrinėje palyginti su Laplaso transformacijos prognoze išreikšta procentine paklaida nuo leistinos generuoti galios

17 lentelė. 2026-4-30 MLP ir Laplaso modelio matematinis palyginimas su realia aktyvia galia

Modelis	MAE, kW	RMSE, kW	Pastaba
Laplasas	2264,908	2555,004	Standartinis suderintas elektrinėje naudojamas modelis, šioje paroje pasirinktas kaip etalonas.
MLP	1607,483	1967,830	Naujesnė automatinės sistemos sugeneruota MLP iteracija pasiekė geresnį rezultatą už Laplaso modelį tiek pagal MAE, tiek pagal RMSE.

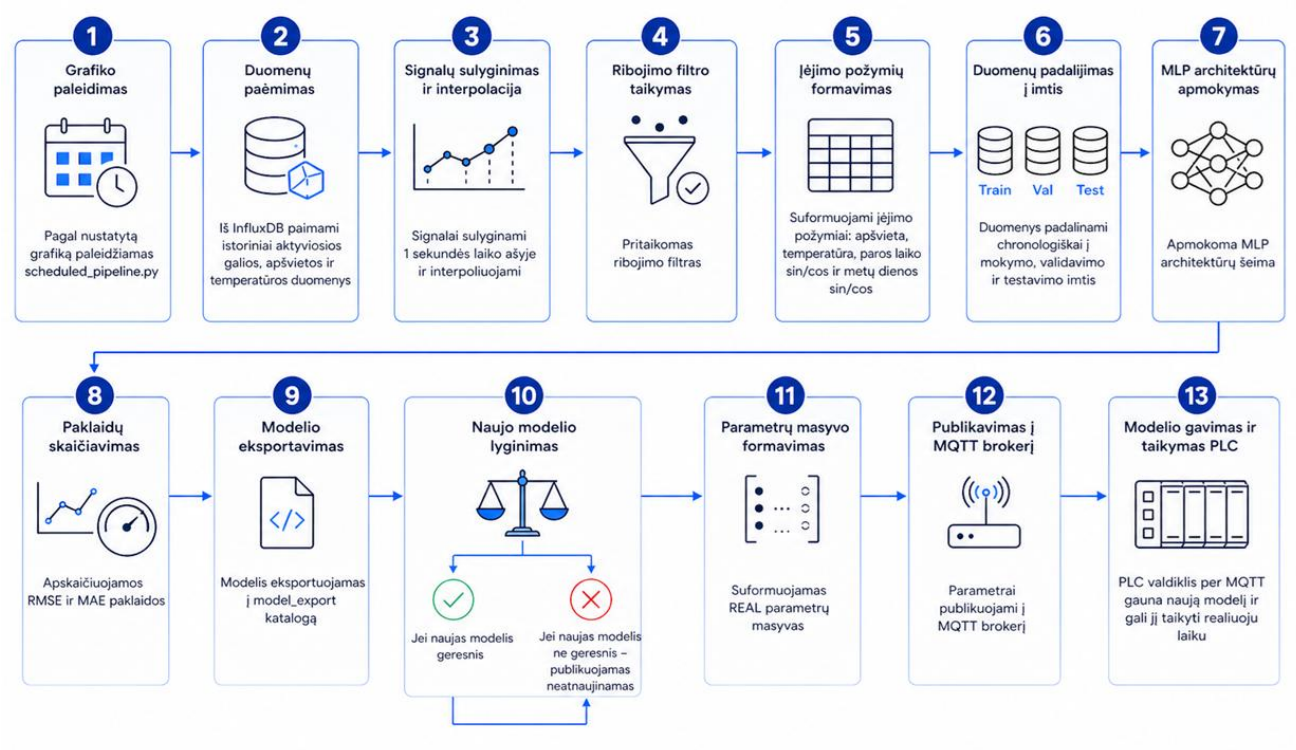
2026-04-19 ir 2026-04-30 dienų palyginimas parodė, kad galutinė dockerizuota MLP mokymo ir publikavimo sistema veikia realiomis elektrinės eksploatavimo sąlygomis. Abiem nagrinėtomis paromis į PLC perduotas MLP modelis pasiekė mažesnes paklaidas nei elektrinėje naudojamas Laplaso transformacijos pagrindu veikiantis modelis.

Taip pat galima pastebėti, kad naujesnės modelio iteracijos išlieka konkurencingos Laplaso transformacijos modeliui. Nors atskirų dienų paklaidos priklauso nuo oro sąlygų, galios svyravimų, debesuotumo ir elektrinės darbo režimo, abiem pateiktais atvejais MLP modelis tiksliau sekė realią aktyviosios galios kreivę. Tai pagrindžia, kad sistema gali periodiškai permokyti modelį su naujais duomenimis, palyginti jį su anksčiau publikuotu modeliu ir į PLC perduoti tik tada, kai naujas modelis atitinka kokybės kriterijus.

Apibendrinant, galutinė dockerizuota sistema pasiekė kelis svarbius tikslus. Pirma, modelio mokymas buvo sėkmingai perkeltas iš lokalaus „PyCharm“ projekto į Linux VM veikiantį konteinerį. Antra, duomenų interpoliavimo logika išsprendė didžiosios dalies eilučių praradimo problemą. Trečia, platesnė MLP architektūrų paieška su dideliu duomenų kiekiu leido reikšmingai pagerinti modelio tikslumą. Ketvirta, sistema automatiškai palygino naują modelį su anksčiau publikuotu ir į PLC kryptį išsiuntė tik geresnį modelį. Realios sistemos bandymai parodė, kad ši logika veikia praktiškai: MLP modelis abiem pateiktomis dienomis pasiekė mažesnes MAE ir RMSE paklaidas nei Laplaso transformacijos pagrindu veikiantis etalonas.

3.5.9. Galutinės sistemos veikimo principas

Galutinė sistema veikia kaip automatinis periodinio MLP modelio mokymo servisas. Šios grandinės svarbiausias privalumas yra tas, kad modelis nebėra statinis. Jis gali būti periodiškai permokomas, naudojant naujausius elektrinės duomenis. Tai leidžia modeliui prisitaikyti prie sezoniškumo, elektrinės pokyčių, modulių degradacijos ar kitų ilgalaikių veiksnių. Kartu išlaikoma saugos logika: į PLC perduodamas tik tas modelis, kuris testavimo imtyje pagerina ankstesnį rezultatą.

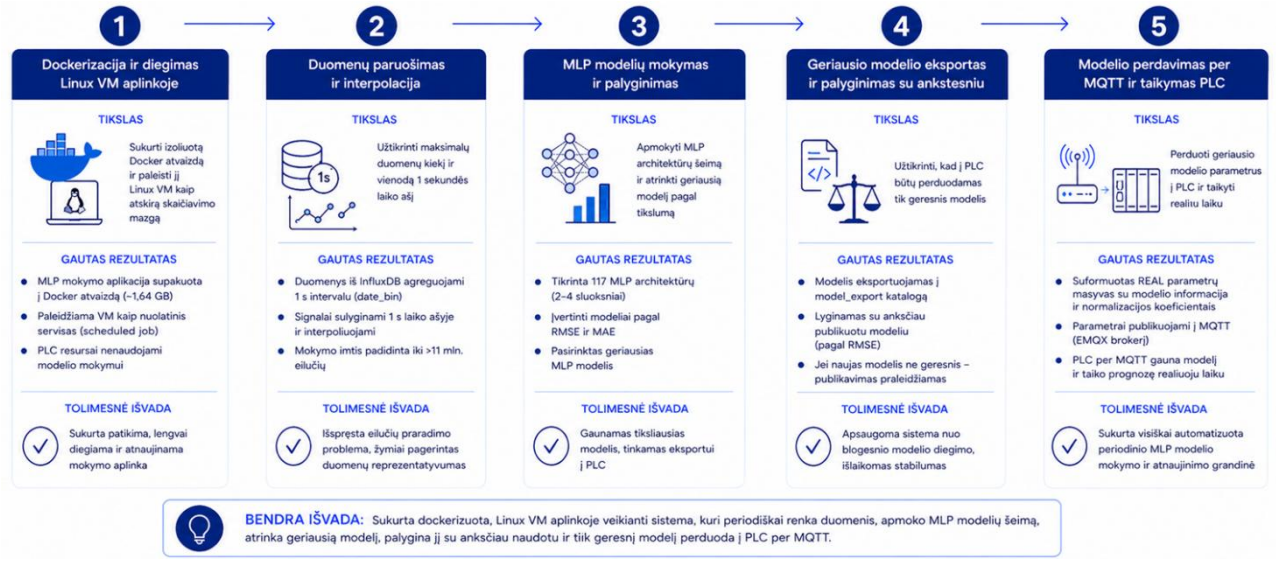


37 pav. Galutinės programos veikimo eiga

3.5.10. Penktojo tyrimo etapo apibendrinimas

Apibendrinant, paskutiniame tyrimo etape buvo pereita nuo modelių palyginimo prie realiai eksploatuojamos automatinės MLP mokymo infrastruktūros. Pirmiausia buvo pasirinktas MLP modelis, nes jis geriausiai atitiko PLC realizavimo, parametrų perdavimo ir skaičiavimo paprastumo reikalavimus. Tada, siekiant optimizuoti „Docker“ aplinką, buvo atsisakyta „PyTorch“ bibliotekos ir pereita prie lengvesnės MLP realizacijos. Toliau buvo išspręsta viena svarbiausių duomenų paruošimo problemų – aktyviosios galios, apšvietos ir temperatūros signalų nesutapimas laike. Interpoliuojant signalus bendroje 1 s laiko ašyje mokymui panaudojamų duomenų kiekis padidėjo nuo šimtų tūkstančių iki daugiau kaip 11 mln. eilučių.

Tarpinis palyginimas parodė, kad viena MLP struktūra su dideliu duomenų kiekiu pasiekė geresnį rezultatą nei 117 architektūrų paieška su praretinta mokymo imtimi. Todėl finaliniame VM bandyme buvo sujungti abu sprendimai: didelis interpoliuotas duomenų rinkinys ir plati MLP architektūrų paieška. Galutinis modelis pasiekė RMSE = 540,331 kW ir MAE = 157,201 kW, o realios sistemos bandymuose MLP modelis abiem nagrinėtomis dienomis pasiekė mažesnes paklaidas nei Laplaso transformacijos modelis. Tai patvirtina, kad sukurta sistema veikia ne tik kaip mokymo algoritmas, bet kaip pilna automatinė grandinė nuo duomenų surinkimo iki modelio taikymo PLC valdiklyje.



38 pav. Paskutinės tyrimo dalies apibendrinimas

3.6. Tiriamosios dalies apibendrinimas

Apibendrinant, tiriamojoje darbo dalyje buvo nuosekliai pereita nuo pirminio MLP modelio prototipo iki realioje elektrinės valdymo sistemoje veikiančios automatinės modelio mokymo ir perdavimo infrastruktūros. Pirmasis tyrimo etapas parodė, kad Python aplinkoje apmokytą MLP modelį galima perkelti į CODESYS ST aplinką ir vykdyti PLC valdiklyje, tačiau pirminės modelio iteracijos tikslumas ir stabilumas dar nebuvo pakankami praktiniam naudojimui.

Toliau buvo sumažintas įėjimo požymių skaičius, atsisakyta statiškų parametrų, sukurta atskira testavimo aplinka ir pradėta sistemingai vertinti modelio paklaida pagal MAE bei RMSE. Šiame etape paaiškėjo, kad modelio tikslumui didelę įtaką turi ne tik neuroninio tinklo struktūra, bet ir duomenų paruošimas. Dėl to buvo įdiegtas galios ribojimo aptikimas pagal apšvietos ir aktyviosios galios viršutinę gaubtinę, o vėliau į modelį įtrauktas temperatūros požymis. Temperatūros analizė parodė, kad šis požymis yra fiziškai reikšmingas, tačiau jo nauda priklauso nuo teisingo signalų sulyginimo.

Platesniame modelių palyginimo etape buvo išbandyti keli prognozavimo modeliai, tokie kaip: MLP, RF, ARIMA, LSTM, RNN, CNN ir hibridinės struktūros. Tyrimas parodė, kad kai kurie modeliai teoriškai gali pasiekti geresnį tikslumą, tačiau jų praktinis taikymas PLC valdiklyje yra ribotas. RF modelis reikalavo per didelio parametrų kiekio, ARIMA buvo jautrus grįžtamajam ryšiui ir ribojimo būsenų neapibrėžtumui, o LSTM ir kiti sekų modeliai buvo per sudėtingi tiesioginei PLC realizacijai. Todėl tolimesniam praktiniam diegimui buvo pasirinktas MLP modelis kaip geriausias tikslumo, paprastumo ir realizavimo galimybių kompromisas.

Sudėtingesni modeliai, tokie kaip LSTM, CNN ar hibridinės architektūros, šiame darbe buvo vertinti kaip teorinis tikslumo potencialas. Jų netaikymas galutiniame PLC diegime nereiškia, kad jie netinkami PV galios prognozavimui, tačiau esamoje PLC vykdymo aplinkoje jų realizavimą riboja parametrų kiekis, sekų apdorojimo poreikis ir didesnė skaičiavimo apkrova. Ateityje, naudojant galingesnę kraštinės kompiuterijos įrangą šie modeliai galėtų būti taikomi plačiau.

Paskutiniame tyrimo etape buvo sukurta dockerizuota automatinio MLP modelio mokymo sistema, veikianti Linux VM aplinkoje. Atsisakius „PyTorch“ bibliotekos ir perėjus prie lengvesnės MLP realizacijos, „Docker“ atvaizdo dydis sumažintas nuo maždaug 20 GB iki apie 1,64 GB. Taip pat

buvo išspręsta viena svarbiausių duomenų paruošimo problemų – aktyviosios galios, apšvietos ir temperatūros signalų nesutapimas laike. Interpoliuojant signalus bendroje 1 s laiko ašyje, mokymui panaudojamų duomenų kiekis padidėjo nuo šimtų tūkstančių iki daugiau kaip 11 mln. eilučių.

Galutinis VM bandymas parodė, kad sujungus didelį interpoliuotų duomenų kiekį ir plačią 117 MLP architektūrų paiešką galima pasiekti reikšmingą tikslumo pagerėjimą. Galutinėje testavimo imtyje modelis pasiekė RMSE = 540,331 kW ir MAE = 157,201 kW. Sistema taip pat automatiškai palygino naują modelį su anksčiau publikuotu ir į PLC perdavė tik geresnį modelį. Tai patvirtino, kad sukurta infrastruktūra gali veikti kaip periodinė modelio permokymo ir atnaujinimo sistema.

Realios elektrinės bandymai parodė, kad į PLC perduotas MLP modelis gali pasiekti mažesnes paklaidas nei iki tol naudotas Laplaso transformacijos pagrindu veikiantis modelis. Taigi tiriamoji dalis patvirtino pagrindinę darbo hipotezę: duomenimis grįstas MLP modelis gali būti ne tik sukurtas ir apmokytas, bet ir praktiškai integruotas į pramoninę elektrinės valdymo sistemą, kurioje jis veikia realiuoju laiku ir gali būti automatiškai atnaujinamas.

4. Tolimesnė kryptis

Atliktas tyrimas parodė, kad realios fotovoltinės elektrinės generuojamos galios prognozavimo tikslumui įtaką daro ne tik pasirinktas modelio tipas, bet ir duomenų kokybė, įėjimo požymių struktūra, debesuotumo dinamika bei praktiniai modelio diegimo apribojimai PLC aplinkoje. Dėl šios priežasties tolimesnis darbo vystymas turėtų būti orientuotas ne vien į sudėtingesnių modelių taikymą, bet ir į informatyvesnių požymių formavimą bei tinkamesnės skaičiavimo architektūros parinkimą.

Pirma yra papildomų dinaminių požymių įtraukimas į modelį. Šiame darbe galutinis MLP modelis daugiausia rėmėsi momentinėmis apšvietos, temperatūros ir laiko požymių reikšmėmis. Tačiau kintamo debesuotumo sąlygomis vien momentinė apšvieta ne visada pakankamai aprašo elektrinės būseną. Todėl ateityje tikslinga formuoti agreguotus požymius, tokius kaip apšvietos slankiojo lango vidurkis, standartinis nuokrypis, minimalios ir maksimalios reikšmės bei apšvietos kitimo greitis. Pavyzdžiui, padidėjęs apšvietos signalo standartinis nuokrypis per trumpą laiko langą gali rodyti debesuotumą ir greitus generacijos svyravimus. Tokie požymiai leistų modeliui geriau atskirti stabilias saulėtas sąlygas nuo greitai kintančių debesuotų režimų.

Antroji kryptis yra elektrinės erdvinės struktūros įvertinimas. Jei elektrinė būtų padalinta į kelias matavimo zonas arba būtų naudojami keli skirtingose parko vietose esantys apšvietos jutikliai, modelis galėtų gauti informaciją ne tik apie bendrą apšvietos lygį, bet ir apie debesų judėjimą per parką. Skirtingų zonų apšvietos pokyčių palyginimas leistų įvertinti, kuri elektrinės dalis pirmiausia paveikiama debesuotumo, o jei papildomai būtų turima vėjo krypties ir greičio informacija, būtų galima numatyti, kaip debesuotumo poveikis persiduos kitoms elektrinės zonoms. Tokia požymių struktūra galėtų pagerinti modelio reakciją staigių apšvietos pokyčių metu.

Trečia yra sudėtingesnių modelių taikymas tinkamesnėje skaičiavimo aplinkoje. Šiame darbe CNN, LSTM, RNN ir hibridinės struktūros buvo vertinamos kaip teorinis tikslumo potencialas, tačiau jų praktinis taikymas PLC aplinkoje yra sudėtingesnis dėl sekų formavimo, vidinės būsenos saugojimo ir didesnio parametrų kiekio. Ateityje, naudojant galingesnį kraštinės kompiuterijos įrenginį šiuos modelius būtų galima vykdyti ne PLC cikle, o atskirame *edge* įrenginyje. Tokiu atveju PLC atliktų tik valdymo ir saugos funkcijas, o sudėtingesnis modelis realiuoju laiku pateiktų apskaičiuotą maksimalios galios įvertį.

Ketvirtoji kryptis yra automatinio modelio vertinimo ir adaptavimo išplėtimas. Šiame darbe jau realizuota logika, kai naujas modelis lyginamas su anksčiau publikuotu modeliu ir pakeičiamas tik tada, kai pasiekia geresnius rezultatus. Ateityje šią logiką būtų galima papildyti atskirais vertinimo režimais: saulėtos dienos, kintamo debesuotumo dienos, žiemos / vasaros laikotarpiai, rytinio kilimo ir vakarinio kritimo intervalai. Tai leistų ne tik vertinti bendrą MAE ar RMSE, bet ir nustatyti, kuriose eksploatacijos sąlygose modelis veikia prasčiausiai.

Apibendrinant, tolimesnis darbo vystymas turėtų būti nukreiptas į debesuotumo dinamikos įtraukimą, agreguotų požymių formavimą, elektrinės zoninės struktūros panaudojimą ir sudėtingesnių modelių vykdymą tinkamesnėje *edge* skaičiavimo aplinkoje. Tai leistų didinti prognozavimo tikslumą, kartu išlaikant šiame darbe suformuotą pagrindinį principą.

Išvados

1. Atlikus literatūrinę analizę nustatyta, kad saulės elektrinių generuojamos galios prognozavimui taikomi fizikiniai, analitiniai, statistiniai, mašininio mokymosi ir hibridiniai modeliai. Literatūroje pabrėžiama, kad šiuolaikinėse PV elektrinėse svarbus ne tik modelio tikslumas, bet ir jo tinkamumas realaus laiko valdymui.
2. Naudojant istorinius realios fotovoltinės elektrinės generacijos, apšvietos, temperatūros ir laiko duomenis buvo sukurti ir palyginti keli prognozavimo modeliai: MLP, RF, ARIMA, LSTM, RNN, CNN ir hibridinės struktūros. Tyrimas parodė, kad nors kai kurie modeliai Python aplinkoje pasiekė gerą tikslumą, praktiniam PLC taikymui tinkamiausias buvo MLP modelis, nes jis suderino pakankamą tikslumą, nedidelį parametrų kiekį ir paprastą realizavimą CODESYS ST aplinkoje.
3. Suprojektuota ir realizuota konteinerizuota programinė infrastruktūra, leidžianti automatiškai surinkti duomenis iš „InfluxDB“, paruošti mokymo imtį, apmokyti MLP, ARIMA, RF ir atrinkti geriausią modelį. Atsisakius „PyTorch“ bibliotekos ir perėjus prie lengvesnės MLP realizacijos, „Docker“ atvaizdo dydis sumažintas nuo maždaug 20 GB iki apie 1,64 GB, todėl sistema tapo tinkamesnė diegimui Linux VM aplinkoje.
4. Parengtas ir įgyvendintas modelio parametrų perdavimo į pramoninį valdiklį sprendimas, paremtas MQTT ryšiu ir bendru REAL tipo parametrų masyvu. PLC pusėje buvo realizuota modelio priėmimo, dekodavimo ir vykdymo logika, todėl apmokytas MLP modelis gali būti automatiškai perduodamas į valdiklį ir naudojamas aktyviosios galios prognozei realiuoju laiku.
5. Eksperimentinis sistemos vertinimas parodė, kad modelio tikslumui didelę įtaką turėjo ne tik pasirinkta architektūra, bet ir duomenų paruošimo kokybė. Tyrimo metu buvo pritaikytas galios ribojimo aptikimo filtras pagal apšvietos ir aktyviosios galios viršutinę gaubtinę, leidęs pašalinti dalį duomenų taškų, kuriuose elektrinė negeneravo maksimalios galimos galios dėl valdymo sistemos ribojimų. Papildomai sulyginus aktyviosios galios, apšvietos ir temperatūros signalus bendroje 1 s laiko ašyje, mokymui panaudojamų duomenų kiekis padidėjo nuo šimtų tūkstančių iki daugiau kaip 11 mln. eilučių, o galutinė MLP sistema testavimo imtyje pasiekė RMSE = 540,331 kW ir MAE = 157,201 kW. Realių elektrinės darbo dienų bandymai parodė, kad į PLC perduotas MLP modelis gali pasiekti mažesnes paklaidas nei šiuo metu taikomas Laplaso transformacijos pagrindu veikiantis modelis, MAE paklaida buvo mažesnė 41,50 %, o RMSE – 35,37 %.

Literatūros sąrašas

1. GEVORGIAN, Vahan. *Highly Accurate Method for Real-Time Active Power Reserve Estimation for Utility-Scale PV Power Plants: Preprint.* /06/14, 2019. Prieiga per: https://www.osti.gov/biblio/1527330..

2. YUAN, Haoyu; TAN, Jin; ZHANG, Yingchen; MURTHY, Samanvitha; YOU, Shutang ir kt. Machine Learning-Based PV Reserve Determination Strategy for Frequency Control on the WECC System. , pp. 1–5.2020-02. Prieiga per: https://ieeexplore.ieee.org/document/9087744..

3. MICHAEL GOGGIN, Grid S. *A Review of Reports on Spanish Blackout Causes and Solutions • Grid Strategies.* -06-24, 2025. Prieiga per: https://gridstrategiesllc.com/project/a-review-of-reports-on-spanish-blackout-causes-and-solutions/..

4. BATZELIS, Efstratios; KAMPITSIS, Georgios ir PAPATHANASSIOU, Stavros. Power Reserves Control for PV Systems with Real-Time MPP Estimation via Curve Fitting. *IEEE Transactions on Sustainable Energy*, vol. 8 (2017), pp. 1269–1280. Prieiga per: https://www.researchgate.net/publication/314011732_Power_Reserves_Control_for_PV_Systems_with_Real-Time_MPP_Estimation_via_Curve_Fitting..

5. ESO. *Elektrinių Atitikties Vertinimo Ir Dokumentų Pateikimo Atmintinė.* , -03-10, 2022. Prieiga per: https://www.eso.lt/download/482780/elektrini%C5%B3%20atitikties%20vertinimo%20ir%20dokument%C5%B3%20pateikimo%20atmintin%C4%97_2022-03-10.docx..

6. ESO. *Natūriniai Bandymai.* [2025. Prieiga per: https://www.eso.lt/duk/naturiniai-bandymai-232..

7. Litgrid. *Perdavimo Tinklo Įrenginių Bandymų Reglamentas.* , 2021. Prieiga per: https://www.litgrid.eu/uploads/files/dir739/dir36/dir1/11_0.php..

8. MAYER, Martin János ir GRÓF, Gyula. Extensive comparison of physical models for photovoltaic power forecasting. *Applied Energy*, vol. 283 (2021), pp. 116239. Prieiga per: https://www.sciencedirect.com/science/article/pii/S0306261920316330..

9. SHI, Ying; SUN, Yue; LIU, Junliang ir DU, Xiong. Model and stability analysis of grid-connected PV system considering the variation of solar irradiance and cell temperature. *International Journal of Electrical Power & Energy Systems*, vol. 132 (2021), pp. 107155. Prieiga per: https://www.sciencedirect.com/science/article/pii/S014206152100394X..

10. ZHANG, Miao; MIAO, Zhixin ir FAN, Lingling. Reduced-Order Analytical Models of Grid-Connected Solar Photovoltaic Systems for Low-Frequency Oscillation Analysis. *IEEE Transactions on Sustainable Energy*, vol. 12 (2021), nr. 3. Prieiga per: https://www.osti.gov/biblio/1848883..

11. MESHRAM, Vipinkumar Shriram; CORTI, Fabio; LOZITO, Gabriele Maria; COSTANZO, Luigi; REATTI, Alberto ir kt. Small-Signal Modeling, Comparative Analysis, and Gain-Scheduled Control of DC–DC Converters in Photovoltaic Applications. *Electronics*, vol. 14 (2025), nr. 21, pp. 4308. Prieiga per: https://www.mdpi.com/2079-9292/14/21/4308..

12. HAN, Peng; HE, Wei; CAO, You; LI, YingMei ir ZHANG, YunYi. Deep belief rule based photovoltaic power forecasting method with interpretability. *Scientific Reports*, vol. 12 (2022), nr. 1, pp. 14467. Prieiga per: https://www.nature.com/articles/s41598-022-18820-6..

13. WANG, Jianbin; LIU, Jiong; CHEN, Bo ir YU, Li. Fast online transfer learning for photovoltaic power prediction. *Journal of the Franklin Institute*, vol. 362 (2025), nr. 17, pp. 108125. Prieiga per: https://www.sciencedirect.com/science/article/pii/S0016003225006179..

14. RODRIGUEZ-ABURTO, César; MONTAÑO-PISFIL, Jorge; SANTOS-MEJÍA, César; MORCILLO-VALDIVIA, Pablo; SOLÍS-FARFÁN, Roberto ir kt. Machine Learning for Photovoltaic Power Forecasting Integrated with Energy Storage Systems: A Scientometric Analysis, Systematic Review, and Meta-Analysis. *Energies*, vol. 18 (2025), nr. 23, pp. 6291. Prieiga per: https://www.mdpi.com/1996-1073/18/23/6291..

15. DIMITROVA-ANGELOVA, Doroteya; GONZÁLEZ-GONZÁLEZ, Juan F.; CARMONA-FERNÁNDEZ, Diego ir JARAMILLO-MORÁN, Miguel A. Photovoltaic Digital Twins: Mathematical Modeling vs. Neural Networks for Energy Management in Smart Buildings. *Applied Sciences*, vol. 15 (2025), nr. 16, pp. 8883. Prieiga per: https://www.mdpi.com/2076-3417/15/16/8883..

16. FERKOUS, Khaled; GUERMOUI, Mawloud; MENAKH, Sarra; BELLAOUR, Abderahmane ir BOULMAIZ, Tayeb. A novel learning approach for short-term photovoltaic power forecasting - A review and case studies. *Engineering Applications of Artificial Intelligence*, vol. 133 (2024), pp. 108502. Prieiga per: https://www.sciencedirect.com/science/article/pii/S0952197624006602..

17. profgriesbauer. *CODESYS Forge - Machine Learning IEC Lib / Wiki / Documentation EN.* [2021-9-15]. Prieiga per: https://forge.codesys.com/lib/ml-iec-lib/wiki/Documentation%20EN/..

18. NIKULINS, Arturs; SUDARS, Kaspars; EDELMERS, Edgars; NAMATEVS, Ivars; OZOLS, Kaspars ir kt. Deep Learning for Wind and Solar Energy Forecasting in Hydrogen Production. *Energies*, vol. 17 (2024), nr. 5, pp. 1053. Prieiga per: https://www.mdpi.com/1996-1073/17/5/1053..

19. Enrique Ramírez. *MLOps at the Edge: Predicting Solar Radiation in Photovoltaic Plants.* -5-5, 2025. Prieiga per: https://www.barbara.tech/blog/mlops-at-the-edge-predicting-solar-radiation-in-photovoltaic-plants..

20. THOMAS, David; ADEBAYO, Bola ir HUSSAIN, Khalid. Digital Twin Modeling for Predictive Maintenance and Fault Detection in Renewable Energy Systems (2025). Prieiga per: https://www.researchgate.net/publication/396751605_Digital_Twin_Modeling_for_Predictive_Maintenance_and_Fault_Detection_in_Renewable_Energy_Systems..

21. DI LEO, Paolo; CIOCIA, Alessandro; MALGAROLI, Gabriele ir SPERTINO, Filippo. Advancements and Challenges in Photovoltaic Power Forecasting: A Comprehensive Review. *Energies*, vol. 18 (2025), nr. 8, pp. 2108. Prieiga per: https://www.mdpi.com/1996-1073/18/8/2108..

22. DIMITROVA-ANGELOVA, Doroteya; GONZÁLEZ-GONZÁLEZ, Juan F.; CARMONA-FERNÁNDEZ, Diego ir JARAMILLO-MORÁN, Miguel A. Photovoltaic Digital Twins: Mathematical Modeling vs. Neural Networks for Energy Management in Smart Buildings. *Applied Sciences*, vol. 15 (2025), nr. 16, pp. 8883. Prieiga per: https://www.mdpi.com/2076-3417/15/16/8883..

23. WAGO. *Edge Controller; 2 X ETHERNET, 2 X USB, 1 X USB-C, HDMI, CAN, DI/DO, RS-232/485, Audio; Control | Controllers, Bus Couplers and I/O | Products.* [2026WAGO. Prieiga per: https://www.wago.com/tr-en/edge-devices/edge-controller/p/752-8303_8000-002..

24. CODESYS. *CODESYS IIoT Libraries SL.* [2026. Prieiga per: https://us.store.codesys.com/codesys-iiot-libraries-sl.html..

25. ALI, Mokhtar; RABEHI, Abdelhalim; SOUAHLIA, Abdelkerim; GUERMOUI, Mawloud; TETA, Ali ir kt. Enhancing PV power forecasting through feature selection and artificial neural networks: a case study. *Scientific Reports*, vol. 15 (2025), nr. 1, pp. 22574. Prieiga per: https://www.nature.com/articles/s41598-025-07038-x..

26. DIMITROVA-ANGELOVA, Doroteya; GONZÁLEZ-GONZÁLEZ, Juan F.; CARMONA-FERNÁNDEZ, Diego ir JARAMILLO-MORÁN, Miguel A. Photovoltaic Digital Twins: Mathematical Modeling vs. Neural Networks for Energy Management in Smart Buildings. *Applied Sciences*, vol. 15 (2025), nr. 16, pp. 8883. Prieiga per: https://www.mdpi.com/2076-3417/15/16/8883..

27. KHOUILI, Oussama; HANINE, Mohamed; LOUZAZNI, Mohamed; FLORES, Miguel Angel López; VILLENA, Eduardo García ir kt. Evaluating the impact of deep learning approaches on solar and photovoltaic power forecasting: A systematic review. *Energy Strategy Reviews*, vol. 59 (2025), pp. 101735. Prieiga per: https://www.sciencedirect.com/science/article/pii/S2211467X25000987..

28. ABDELSATTAR, Montaser; ABDELMOETY, Ahmed ir EMAD-ELDEEN, Ahmed. Advanced machine learning techniques for predicting power generation and fault detection in solar photovoltaic systems. *Neural Computing and Applications*, vol. 37 (2025), nr. 15, pp. 8825–8844. Prieiga per: https://doi.org/10.1007/s00521-025-11035-6..

29. SALTOS, Joan M.; INTRIAGO CEDEÑO, M. Gabriela; BALDERRAMO VELEZ, Ney R.; LEÓN, Germán T. Ramos ir CANO-ORTEGA, A. Hybrid AI Models for Short-Term Photovoltaic Forecasting: A Systematic Review of Architectures, Performance, and Deployment Challenges. *Sensors*, vol. 26 (2026), nr. 6, pp. 1793. Prieiga per: https://www.mdpi.com/1424-8220/26/6/1793..

30. FAN, Yuanliang; WU, Han; LIN, Jianli; LI, Zewen; LI, Lingfei ir kt. A distributed photovoltaic short-term power forecasting model based on lightweight AI for edge computing. *Journal of Physics: Conference Series*, vol. 2876 (2024), nr. 1, pp. 012050. Prieiga per: https://doi.org/10.1088/1742-6596/2876/1/012050..

31. HANSEN, Clifford W.; HOLMGREN, William F.; TUOHY, Aidan; SHARP, Justin; LORENZO, Antonio T. ir kt. The Solar Forecast Arbiter: An Open Source Evaluation Framework for Solar Forecasting. , pp. 2452–2457.2019-06. Prieiga per: https://ieeexplore.ieee.org/document/8980713..

Informacijos šaltinių sąrašas

1. pav. 1 https://tspi.lt/image/cache/catalog/homepage/tspi-1-cr-1200x960.png
2. pav. 2 https://tspi.lt/image/cache/catalog/Categories/Desktop%20ir%20mobile/property-lt-cr-1200x960.png

Priedai

1 priedas. Pirmosios *CODESYS* ST modelio adaptacijos kodas

Priede pateikiama pirmosios MLP modelio realizacijos CODESYS ST aplinkoje struktūra. Kadangi pilni neuroninio tinklo svorių, poslinkių ir normalizavimo koeficientų masyvai užima daug vietos ir apsunkina kodo skaitomumą, priede jie pateikiami kaip struktūriniai masyvai, nurodant jų paskirtį. Faktinės reikšmės buvo rankiniu būdu eksportuojamos iš Python aplinkoje apmokyto modelio ir įkeliamos į CODESYS programą. Toks pateikimas leidžia aiškiai parodyti modelio vykdymo logiką: įėjimų formavimą, normalizavimą, sluoksnių skaičiavimą, ReLU aktyvacijas ir galutinės aktyviosios galios reikšmės denormalizavimą.

```
(*
    Funkcinis blokas: NN_Model
    Įėjimo požymiai:
        X[1] – saulės apšvieta, W/m²
        X[2] – modulio / aplinkos temperatūra, °C
        X[3] – paros laiko sinusinė dedamoji
        X[4] – paros laiko kosinusinė dedamoji
        X[5] – metų dienos sinusinė dedamoji
        X[6] – metų dienos kosinusinė dedamoji
        X[7] – azimuto sinusinė dedamoji
        X[8] – azimuto kosinusinė dedamoji
        X[9] – požymis, nurodantis ar elektrinė turi sekimo sistemą
    Išėjimas:
        P_Pred_kW – prognozuojama aktyvioji galia, kW
*)

FUNCTION_BLOCK NN_Model
VAR_INPUT
    Irradiance  : REAL;          (* Saulės apšvieta, W/m² *)
    Temperature : REAL;          (* Temperatūra, °C *)
    Azimuth     : REAL := 180.0; (* Saulės / elektrinės azimutas,
laipsniais *)
    HasTrackers : REAL := 0.0;   (* 0.0 – be sekimo sistemos, 1.0 –
su sekimo sistema *)
END_VAR

VAR_OUTPUT
    P_Pred_kW : REAL;            (* Prognozuojama aktyvioji galia,
kW *)
END_VAR

VAR CONSTANT
    PI : REAL := 3.14159265;
    N_INPUT  : INT := 9;
    N_H1     : INT := 16;
```

```
    N_H2     : INT := 8;
END_VAR

VAR
    X_Mean   : ARRAY[1..9] OF REAL := [ (* x1_mean, x2_mean, ..., x9_mean *) ];
    X_Scale : ARRAY[1..9] OF REAL := [ (* x1_scale, x2_scale, ..., x9_scale *) ];

    Y_Mean  : REAL := (* y_mean *);
    Y_Scale : REAL := (* y_scale *);

    FC1_W : ARRAY[1..16] OF ARRAY[1..9] OF REAL := [ (* 16 eilučių po 9 svorius *) ];
    FC1_B : ARRAY[1..16] OF REAL := [(* 16 poslinkių *)];

    FC2_W : ARRAY[1..8] OF ARRAY[1..16] OF REAL := [ (* 8 eilutės po 16 svorių *) ];
    FC2_B : ARRAY[1..8] OF REAL := [ (* 8 poslinkiai *) ];

    FC3_W : ARRAY[1..8] OF REAL := [ (* 8 svoriai *) ];
    FC3_B : REAL := (* išėjimo sluoksnio poslinkis *);

    (* Tarpiniai skaičiavimo kintamieji *)
    X_raw    : ARRAY[1..9] OF REAL;
    X_scaled : ARRAY[1..9] OF REAL;
    H1       : ARRAY[1..16] OF REAL;
    H2       : ARRAY[1..8] OF REAL;
    i   : INT;
    j   : INT;
    sum : REAL;

    (* Laiko požymių skaičiavimui naudojami kintamieji *)
    REAL_H : REAL;   (* Paros valanda realiuoju formatu, pvz. 13.5 *)
    REAL_D : REAL;   (* Metų diena, 1-365/366 *)
    today          : DATE;
    daysSinceEpoch : DINT;
    secSinceEpoch  : UDINT;
    tmpDays        : DINT;
    year           : INT;
    daysInYear     : INT;
END_VAR
```

```
(*Iš PLC sisteminio UTC laiko apskaičiuojama reali paros valanda*)
```

```
REAL_H :=
    UDINT_TO_REAL(TO_DWORD(TimeNow_UTC0)    /    TO_DWORD(T#1H))    +
UDINT_TO_REAL((TO_DWORD(TimeNow_UTC0)    MOD    TO_DWORD(T#1H))    /
TO_DWORD(T#1M)) / 60.0;


(*Iš PLC datos apskaičiuojama kelinta metų diena yra einamoji data.*)
today := DateNow_UTC0;

secSinceEpoch  := TO_UDINT(today) - TO_UDINT(DATE#1970-01-01);
daysSinceEpoch := TO_DINT(secSinceEpoch / 86400);

tmpDays := daysSinceEpoch;
year    := 1970;

WHILE TRUE DO
    (* Keliamųjų metų patikra *)
    IF ((year MOD 4 = 0) AND (year MOD 100 <> 0)) OR (year MOD 400
= 0) THEN
        daysInYear := 366;
    ELSE
        daysInYear := 365;
    END_IF;

    IF tmpDays >= daysInYear THEN
        tmpDays := tmpDays - daysInYear;
        year    := year + 1;
    ELSE
        EXIT;
    END_IF;
END_WHILE;

REAL_D := TO_REAL(tmpDays + 1);


(*Įėjimo vektorių sudaro tiesioginiai elektrinės matavimai ir
papildomi cikliniai požymiai*)
X_raw[1] := Irradiance;
X_raw[2] := Temperature;

(* Paros laiko kodavimas *)
X_raw[3] := SIN(2.0 * PI * REAL_H / 24.0);
X_raw[4] := COS(2.0 * PI * REAL_H / 24.0);

(* Metų dienos kodavimas *)
X_raw[5] := SIN(2.0 * PI * REAL_D / 365.0);
```

```
X_raw[6] := COS(2.0 * PI * REAL_D / 365.0);

(* Azimuto kodavimas *)
X_raw[7] := SIN(Azimuth * PI / 180.0);
X_raw[8] := COS(Azimuth * PI / 180.0);

(* Sekimo sistemos požymis *)
X_raw[9] := HasTrackers;


(*Kiekvienas įėjimas normalizuojamas naudojant mokymo metu gautus vidurkius ir skales*)
FOR i := 1 TO 9 DO
    IF X_Scale[i] <> 0.0 THEN
        X_scaled[i] := (X_raw[i] - X_Mean[i]) / X_Scale[i];
    ELSE
        (* Apsauga nuo dalybos iš nulio *)
        X_scaled[i] := X_raw[i] - X_Mean[i];
    END_IF;
END_FOR;


(*Kiekvienam neuronui apskaičiuojama svorinė suma, po to taikoma ReLU aktyvacijos funkcija*)
FOR i := 1 TO 16 DO
    sum := FC1_B[i];

    FOR j := 1 TO 9 DO
        sum := sum + FC1_W[i][j] * X_scaled[j];
    END_FOR;

    IF sum > 0.0 THEN
        H1[i] := sum;
    ELSE
        H1[i] := 0.0;
    END_IF;
END_FOR;


(*Šiame sluoksnyje naudojama pirmojo sluoksnio išvestis H1*)
FOR i := 1 TO 8 DO
    sum := FC2_B[i];

    FOR j := 1 TO 16 DO
        sum := sum + FC2_W[i][j] * H1[j];
    END_FOR;
```

```
    IF sum > 0.0 THEN
        H2[i] := sum;
    ELSE
        H2[i] := 0.0;
    END_IF;
END_FOR;


(*Išėjimo sluoksnis yra tiesinis. Aktyvacijos funkcija čia
netaikoma, nes prognozuojama tolydi aktyviosios galios reikšmė*)
sum := FC3_B;

FOR j := 1 TO 8 DO
    sum := sum + FC3_W[j] * H2[j];
END_FOR;


(*Modelio išvestis grąžinama iš normalizuotos skalės į fizikinę
aktyviosios galios reikšmę kilovatais*)
P_Pred_kW := sum * Y_Scale + Y_Mean;
```

2 priedas. Universali modelio realizacija *CODESYS* ST aplinkoje

Priede pateikiama patobulinta modelio realizacijos *CODESYS* ST aplinkoje struktūra. Skirtingai nei pirmojoje versijoje, šiame funkciniame bloke modelio parametrai nėra įrašomi kaip atskiri kiekvieno sluoksnio svorių ir poslinkių masyvai. Vietoje to naudojamas bendras parametrų masyvas *„NN_Params“*, kuriame saugomi modelio normalizavimo koeficientai, struktūros informacija ir pasirinkto modelio parametrai. Toks sprendimas leidžia tą patį funkcinį bloką naudoti kelių tipų modeliams: MLP, RF arba ARIMA. Modelio tipas parenkamas pagal kintamąjį *„NN_ModelType“*.

```
(*Modelio pastovios vertės*)
VAR PERSISTENT
    NN_ModelType   : BYTE := 3;       (* 1 = MLP, 2 = RF, 3 = ARIMA
*)
    NN_ParamCount : UINT := 32;

    (* Bendras modelio parametrų masyvas:
       normalizavimo koeficientai, modelio struktūra ir svoriai /
koeficientai. *)
    NN_Params : ARRAY[1..1100] OF REAL := [];

    (* ARIMA modeliui naudojama išėjimo ir liekanų istorija. *)
    NN_ARIMA_Y_Hist : ARRAY[1..16] OF REAL;
    NN_ARIMA_E_Hist : ARRAY[1..16] OF REAL;
END_VAR
```

```
(*Funkcinio bloko iškvietimas realioje programoje*)
NN_Model(
    Irradiance  := Irradiance_mean,
    Temperature := Temperature_mean,
    P_Pred_kW   => modeled_load,
    P_now       := SolarGridValues.ActivePower,
    Cur_now             :=  TotalSunMaxLimit_ProcFinal  /  100  *
TotalSunMaxInverterPower
);
```

Šioje realizacijoje *P_now* naudojamas kaip esama išmatuota aktyvioji galia, o *Cur_now* – kaip tuo metu galiojanti galios ribojimo reikšmė. Šie signalai reikalingi ARIMA modelio šakos grįžtamajam ryšiui, kad istorija nebūtų atnaujinama ribotos galios matavimais.

```
(* Funkcinis blokas: NN_Model
    Modelio tipas:
        NN_ModelType = 1 – MLP modelis
        NN_ModelType = 2 – Random Forest modelis
        NN_ModelType = 3 – ARIMA / ARX modelis
    Įėjimo požymiai:
        X[1] – saulės apšvieta, W/m²
        X[2] – modulio / aplinkos temperatūra, °C
        X[3] – paros laiko sinusinė dedamoji
        X[4] – paros laiko kosinusinė dedamoji
        X[5] – metų dienos sinusinė dedamoji
        X[6] – metų dienos kosinusinė dedamoji
    Papildomi įėjimai:
        P_now   – reali aktyvioji galia, kW
        Cur_now – tuo metu galiojanti galios ribojimo reikšmė, kW
    Išėjimas:
        P_Pred_kW – prognozuojama aktyvioji galia, kW
*)

FUNCTION_BLOCK NN_Model
VAR_INPUT
    Irradiance  : REAL;          (* Saulės apšvieta, W/m² *)
    Temperature : REAL;          (* Temperatūra, °C *)
    P_now       : REAL;          (* Esama reali aktyvioji galia, kW
*)
    Cur_now     : REAL;          (* Esama ribojimo reikšmė, kW *)
END_VAR

VAR_OUTPUT
```

```
    P_Pred_kW : REAL;              (* Prognozuojama aktyvioji galia,
kW *)
END_VAR

VAR
    X_raw    : ARRAY[1..6] OF REAL;   (* Neapdorotas įėjimo vektorius
*)
    X_scaled : ARRAY[1..6] OF REAL;   (* Normalizuotas įėjimo
vektorius *)

    A_Prev : ARRAY[1..64] OF REAL;    (* Ankstesnio sluoksnio
reikšmės *)
    A_Curr : ARRAY[1..64] OF REAL;    (* Einamojo sluoksnio reikšmės
*)

    i, j, k : INT;
    idx     : INT;
    inCount : INT;
    outCount : INT;
    layerCount : INT;
    sum : REAL;

    REAL_H : REAL;   (* Paros valanda realiuoju formatu *)
    REAL_D : REAL;   (* Metų diena *)

    today          : DATE;
    daysSinceEpoch : DINT;
    secSinceEpoch  : UDINT;
    tmpDays        : DINT;
    year           : INT;
    daysInYear     : INT;

    xMeanStart   : INT;   (* Įėjimų vidurkių pradžia NN_Params masyve
*)
    xScaleStart  : INT;   (* Įėjimų skalių pradžia NN_Params masyve
*)
    payloadStart : INT;   (* Modelio parametrų pradžia NN_Params
masyve *)

    (* Random Forest skaičiavimui naudojami kintamieji *)
    treeCount : INT;
    nodeCount : INT;
    treeIdx   : INT;
    nodeIdx   : INT;
    nodeType  : INT;
    featIdx   : INT;
```

```
    leftIdx   : INT;
    rightIdx  : INT;
    threshold : REAL;
    leafValue : REAL;
    treeSum   : REAL;

    (* ARIMA / ARX modelio kintamieji *)
    p : INT;
    d : INT;
    q : INT;
    exogCount : INT;
    constTerm : REAL;
    yhat      : REAL;

    predScaled     : REAL;
    y_meas_scaled  : REAL;
    resid_scaled   : REAL;
    useMeasurement : BOOL;
    curtailedLikely : BOOL;
    curMarginAbs   : REAL;
    curMarginRel   : REAL;
    limitBand      : REAL;
END_VAR
```

```
(* Pradinė validacija: jei modelio tipas nenurodytas arba parametrų
per mažai,
   modelio išvestis nustatoma į nulį. *)
IF NN_ModelType = 0 THEN
    P_Pred_kW := 0.0;
    RETURN;
END_IF;

IF NN_ParamCount < 24 THEN
    P_Pred_kW := 0.0;
    RETURN;
END_IF;
(* Iš PLC sisteminio UTC laiko apskaičiuojama reali paros valanda.
*)
REAL_H :=
    UDINT_TO_REAL(TO_DWORD(TimeNow_UTC0) / TO_DWORD(T#1H))
    + UDINT_TO_REAL((TO_DWORD(TimeNow_UTC0) MOD TO_DWORD(T#1H)) /
TO_DWORD(T#1M)) / 60.0;
(* Iš PLC datos apskaičiuojama einamoji metų diena. *)
today := DateNow_UTC0;
```

```
secSinceEpoch  := TO_UDINT(today) - TO_UDINT(DATE#1970-01-01);
daysSinceEpoch := TO_DINT(secSinceEpoch / 86400);

tmpDays := daysSinceEpoch;
year    := 1970;

WHILE TRUE DO
    IF ((year MOD 4 = 0) AND (year MOD 100 <> 0)) OR (year MOD 400
= 0) THEN
        daysInYear := 366;
    ELSE
        daysInYear := 365;
    END_IF;

    IF tmpDays >= daysInYear THEN
        tmpDays := tmpDays - daysInYear;
        year    := year + 1;
    ELSE
        EXIT;
    END_IF;
END_WHILE;

REAL_D := TO_REAL(tmpDays + 1);
(* Sudaromas šešių įėjimų vektorius. *)
X_raw[1] := Irradiance;
X_raw[2] := Temperature;
X_raw[3] := SIN(2.0 * 3.14159265 * REAL_H / 24.0);
X_raw[4] := COS(2.0 * 3.14159265 * REAL_H / 24.0);
X_raw[5] := SIN(2.0 * 3.14159265 * REAL_D / 365.0);
X_raw[6] := COS(2.0 * 3.14159265 * REAL_D / 365.0);
(* Iš bendro parametrų masyvo nuskaitomas modelio įėjimų skaičius.
*)
inCount := REAL_TO_INT(NN_Params[4]);

IF inCount <> 6 THEN
    P_Pred_kW := 0.0;
    RETURN;
END_IF;

xMeanStart   := 6;
xScaleStart  := xMeanStart + inCount;
payloadStart := xScaleStart + inCount;
(* Įėjimai normalizuojami naudojant Python aplinkoje apskaičiuotus
   vidurkius ir mastelio koeficientus. *)
FOR i := 1 TO inCount DO
    IF NN_Params[xScaleStart + i - 1] <> 0.0 THEN
```

```
            X_scaled[i] :=
                (X_raw[i] - NN_Params[xMeanStart + i - 1])
                / NN_Params[xScaleStart + i - 1];
        ELSE
            X_scaled[i] := X_raw[i] - NN_Params[xMeanStart + i - 1];
        END_IF;
    END_FOR;
    CASE NN_ModelType OF
    1:  (* MLP modelio vykdymas *)

        (* Į A_Prev masyvą įrašomas normalizuotas įėjimo vektorius. *)
        FOR i := 1 TO 64 DO
            A_Prev[i] := 0.0;
            A_Curr[i] := 0.0;
        END_FOR;

        FOR i := 1 TO inCount DO
            A_Prev[i] := X_scaled[i];
        END_FOR;

        (* Iš NN_Params masyvo nuskaitomas sluoksnių skaičius ir jų
    dydžiai. *)
        idx := payloadStart;
        layerCount := REAL_TO_INT(NN_Params[idx]);
        idx := idx + 1;

        IF (layerCount < 1) OR (layerCount > 8) THEN
            P_Pred_kW := 0.0;
            RETURN;
        END_IF;

        idx := payloadStart + 1;

        FOR i := 1 TO layerCount DO
            A_Curr[i] := NN_Params[idx];
            idx := idx + 1;
        END_FOR;

        inCount := 6;

        (* Kiekvienas MLP sluoksnis skaičiuojamas iš bendro parametrų
    masyvo:
           pirmiausia taikoma svorinė suma, tada poslinkis, o
    paslėptuose
           sluoksniuose – ReLU aktyvacijos funkcija. *)
        FOR k := 1 TO layerCount DO
```

```
            outCount := REAL_TO_INT(A_Curr[k]);

            FOR i := 1 TO outCount DO
                sum := 0.0;

                FOR j := 1 TO inCount DO
                    sum := sum + NN_Params[idx] * A_Prev[j];
                    idx := idx + 1;
                END_FOR;

                sum := sum + NN_Params[idx];
                idx := idx + 1;

                IF k < layerCount THEN
                    IF sum > 0.0 THEN
                        A_Curr[32 + i] := sum;
                    ELSE
                        A_Curr[32 + i] := 0.0;
                    END_IF;
                ELSE
                    A_Curr[32 + i] := sum;
                END_IF;
            END_FOR;

            FOR i := 1 TO 64 DO
                A_Prev[i] := 0.0;
            END_FOR;

            FOR i := 1 TO outCount DO
                A_Prev[i] := A_Curr[32 + i];
                A_Curr[32 + i] := 0.0;
            END_FOR;

            inCount := outCount;
        END_FOR;

        (* Išvestis grąžinama iš normalizuotos skalės į kW. *)
        predScaled := A_Prev[1];
        P_Pred_kW := predScaled * NN_Params[3] + NN_Params[2];
2:      (* Random Forest modelio vykdymas *)

        (* Kiekvienas medis skaitomas iš NN_Params masyvo.
           Pagal požymio reikšmę ir slenkstį pasirenkamas kairysis arba
dešinysis mazgas,
           kol pasiekiamas lapas. Galutinė prognozė yra visų medžių
vidurkis. *)
```

```
    idx := payloadStart;
    treeCount := REAL_TO_INT(NN_Params[idx]);
    idx := idx + 1;

    IF treeCount < 1 THEN
        P_Pred_kW := 0.0;
        RETURN;
    END_IF;

    treeSum := 0.0;

    FOR treeIdx := 1 TO treeCount DO
        nodeCount := REAL_TO_INT(NN_Params[idx]);
        idx := idx + 1;

        nodeIdx := 1;

        WHILE TRUE DO
            j := idx + (nodeIdx - 1) * 6;

            nodeType  := REAL_TO_INT(NN_Params[j]);
            featIdx   := REAL_TO_INT(NN_Params[j + 1]);
            threshold := NN_Params[j + 2];
            leftIdx   := REAL_TO_INT(NN_Params[j + 3]);
            rightIdx  := REAL_TO_INT(NN_Params[j + 4]);
            leafValue := NN_Params[j + 5];

            IF nodeType = 1 THEN
                treeSum := treeSum + leafValue;
                EXIT;
            END_IF;

            IF X_scaled[featIdx] <= threshold THEN
                nodeIdx := leftIdx;
            ELSE
                nodeIdx := rightIdx;
            END_IF;
        END_WHILE;

        idx := idx + nodeCount * 6;
    END_FOR;

    predScaled := treeSum / TO_REAL(treeCount);
    P_Pred_kW := predScaled * NN_Params[3] + NN_Params[2];
3:  (* ARIMA / ARX modelio vykdymas su grįžtamuoju ryšiu *)
```

```
    (* Iš NN_Params masyvo nuskaitomi ARIMA modelio parametrai:
       p – autoregresijos eilė,
       d – integravimo eilė,
       q – slankiojo vidurkio eilė,
       constTerm – pastovusis narys,
       exogCount – išorinių požymių skaičius. *)
    idx := payloadStart;

    p := REAL_TO_INT(NN_Params[idx]); idx := idx + 1;
    d := REAL_TO_INT(NN_Params[idx]); idx := idx + 1;
    q := REAL_TO_INT(NN_Params[idx]); idx := idx + 1;
    constTerm := NN_Params[idx]; idx := idx + 1;
    exogCount := REAL_TO_INT(NN_Params[idx]); idx := idx + 1;

    IF d <> 0 THEN
        P_Pred_kW := 0.0;
        RETURN;
    END_IF;

    yhat := constTerm;

    (* AR dalis – naudojama ankstesnių prognozių / matavimų istorija.
*)
    FOR i := 1 TO p DO
        yhat := yhat + NN_Params[idx] * NN_ARIMA_Y_Hist[i];
        idx := idx + 1;
    END_FOR;

    (* MA dalis – naudojama ankstesnių liekanų istorija. *)
    FOR i := 1 TO q DO
        yhat := yhat + NN_Params[idx] * NN_ARIMA_E_Hist[i];
        idx := idx + 1;
    END_FOR;

    (* Išoriniai požymiai – apšvieta, temperatūra ir cikliniai laiko
požymiai. *)
    FOR i := 1 TO exogCount DO
        yhat := yhat + NN_Params[idx] * X_scaled[i];
        idx := idx + 1;
    END_FOR;

    predScaled := yhat;
    P_Pred_kW := predScaled * NN_Params[3] + NN_Params[2];
    (* Ribojimo aptikimo logika.
       Jei reali aktyvioji galia yra arti ribojimo reikšmės,
laikoma, kad
```

```
       elektrinė tikėtinai ribojama. Tokiu atveju matavimas
nenaudojamas
       ARIMA istorijai atnaujinti, kad modelis nesimokytų iš
dirbtinai
       sumažintos galios. *)
    curMarginAbs := 50.0;
    curMarginRel := 0.05;

    limitBand := curMarginAbs;

    IF (Cur_now * curMarginRel) > limitBand THEN
        limitBand := Cur_now * curMarginRel;
    END_IF;

    curtailedLikely := FALSE;

    IF Cur_now > 0.0 THEN
        IF P_now >= (Cur_now - limitBand) THEN
            curtailedLikely := TRUE;
        END_IF;
    END_IF;

    useMeasurement := NOT curtailedLikely;
    (* Išmatuota aktyvioji galia normalizuojama pagal išėjimo skalę.
*)
    IF NN_Params[3] <> 0.0 THEN
        y_meas_scaled := (P_now - NN_Params[2]) / NN_Params[3];
    ELSE
        y_meas_scaled := 0.0;
    END_IF;

    (* Liekanos skaičiuojamos tik tada, kai matavimas laikomas
patikimu. *)
    IF useMeasurement THEN
        resid_scaled := y_meas_scaled - predScaled;
    ELSE
        resid_scaled := 0.0;
    END_IF;

    (* Atnaujinama ARIMA istorija. *)
    FOR i := 16 TO 2 BY -1 DO
        NN_ARIMA_Y_Hist[i] := NN_ARIMA_Y_Hist[i - 1];
        NN_ARIMA_E_Hist[i] := NN_ARIMA_E_Hist[i - 1];
    END_FOR;

    IF useMeasurement THEN
```

```
            NN_ARIMA_Y_Hist[1] := y_meas_scaled;
            NN_ARIMA_E_Hist[1] := resid_scaled;
        ELSE
            (* Ribojimo metu istorija neatnaujinama realiu ribotu
matavimu. *)
            NN_ARIMA_Y_Hist[1] := predScaled;
            NN_ARIMA_E_Hist[1] := 0.0;
        END_IF;
END_CASE;
(* Galutinė prognozė apribojama fiziškai leistinu elektrinės galios
intervalu. *)
P_Pred_kW := MAX(MIN(P_Pred_kW, TotalSunMaxPlantPower), 0);
```

3 priedas. Modelio parametrų gavimas per MQTT ir įkėlimas į PLC masyvą

Priede pateikiama MQTT logika, skirta neuroninio tinklo arba kito prognozavimo modelio parametrams perduoti į PLC valdiklį. Šiame sprendime modelio parametrai nėra rankiniu būdu įrašomi į programą, o gaunami per MQTT pranešimą. Tam naudojama *„WAGO“* MQTT biblioteka, JSON formato duomenų nuskaitymas ir atskiras JSON apdorojimo funkcinis blokas.

```
FUNCTION_BLOCK FbMQTTread_NN

VAR_INPUT
    En       : BOOL;     (* Įjungia MQTT ryšį *)
    sTopic   : STRING;   (* Elektrinės topic pavadinimas *)
END_VAR

VAR
    mqttSubscribe:
WagoAppCloud.FbSubscribeMQTT_2(eConnection:=econnectionid.Connecti
on1);
    aPayloadData  : ARRAY[0..250000] OF BYTE;  (* Gauta žinutė*)
    aPayloadLen   : DWORD;                     (* Gautos žinutės
ilgis *)
    xDataReceived : BOOL;                      (* Ar gauti nauji
duomenys *)
    jsonC         : WagoAppCloud.FbGetSingleKeyValue;
    JsonHandler   : NNJsonHandler;  (* JSON apdorojimo blokas *)
    JsonParser    : WagoAppJson.FbJSON_Sax_Parser(JsonHandler);
    StartParsing  : BOOL;  (* JSON analizės paleidimas *)
    ParserDone    : BOOL;  (* JSON analizės pabaiga *)
END_VAR
VAR_IN_OUT
    NN_ModelType  : BYTE;         (* Modelio tipas *)
    NN_ParamCount : UINT;         (* Parametrų kiekis *)
    NN_Params     : ARRAY [*] OF REAL;  (* Parametrų masyvas *)
```

```
END_VAR
```

```
(* MQTT prenumeravimas *)
mqttSubscribe(
    xSubscribe    := En,
    sTopic        := CONCAT('plants/NN_params/', sTopic),
    aPayloadData  := aPayloadData,
    xDataReceived => xDataReceived,
    dwRxNBytes    => aPayloadLen
);
IF En THEN
    (* Jei gautas naujas MQTT pranešimas – pradedamas JSON duomenų
atrinkimas *)
    IF xDataReceived THEN
        StartParsing := TRUE;
    END_IF;
    IF StartParsing THEN
        JsonParser.RegisterParserListener(JsonHandler);

        (* Apdorojimo funkcijai perduodami išoriniai kintamieji *)
        JsonHandler(
            NN_ModelType  := NN_ModelType,
            NN_ParamCount := NN_ParamCount,
            NN_Params     := NN_Params
        );

        JsonParser(
            pData         := ADR(aPayloadData),
            udiSizeData   := aPayloadLen,
            sFileName     := ,
            xTrigger      := StartParsing,
            oStatus       => ,
            xDone         => ParserDone,
            xError        => ,
            udiValueCount=> ,
            iChunksCompleted =>
        );
    END_IF;

    IF ParserDone OR JsonParser.xError THEN
        ParserDone := FALSE;
    END_IF;

END_IF;
```

„JsonParser“ funkcijos pagrindiniai metodai:

„JSON_Value_detected“ metodas:

```
(* Jei skaitomas masyvas – reikšmės pildomos į NN_Params *)
IF ArrayStart THEN
    NN_Params[ArrayValue_nr] := STRING_TO_REAL(sJSON_Value);
    ArrayValue_nr := ArrayValue_nr + 1;

(* Modelio tipo nuskaitymas *)
ELSIF currentKey = 'model_type' THEN
    NN_ModelType := STRING_TO_BYTE(sJSON_Value);

(* Parametrų kiekio nuskaitymas *)
ELSIF currentKey = 'param_count' THEN
    NN_ParamCount := STRING_TO_UINT(sJSON_Value);
END_IF;
```

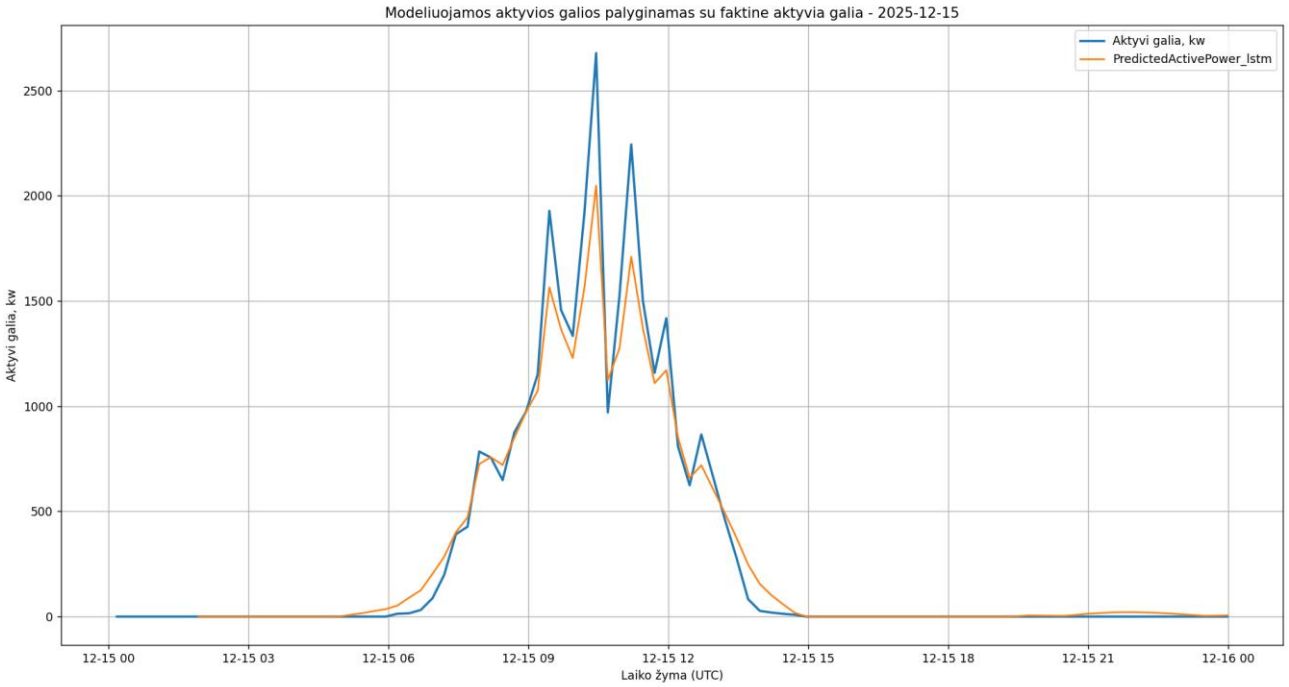
„Json_ArrayClosed“ metodas:

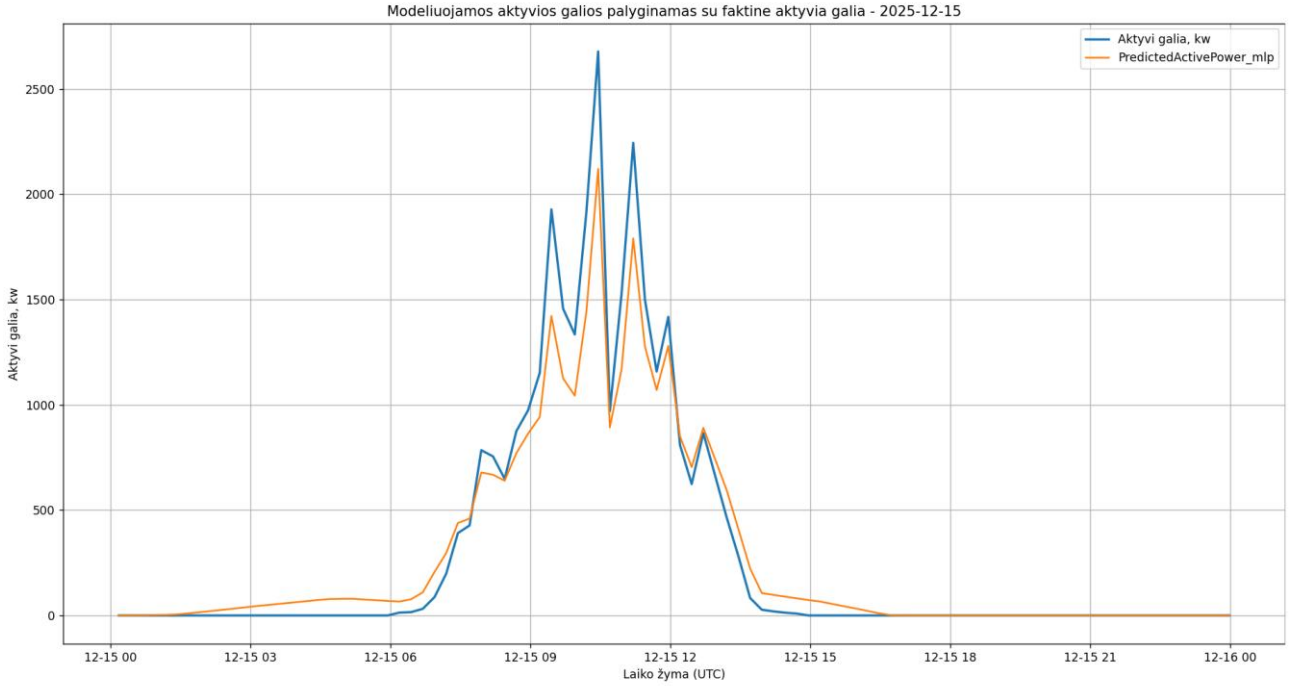
```
(* Aptikus JSON masyvo pabaigą *)
ArrayStart := FALSE;
```

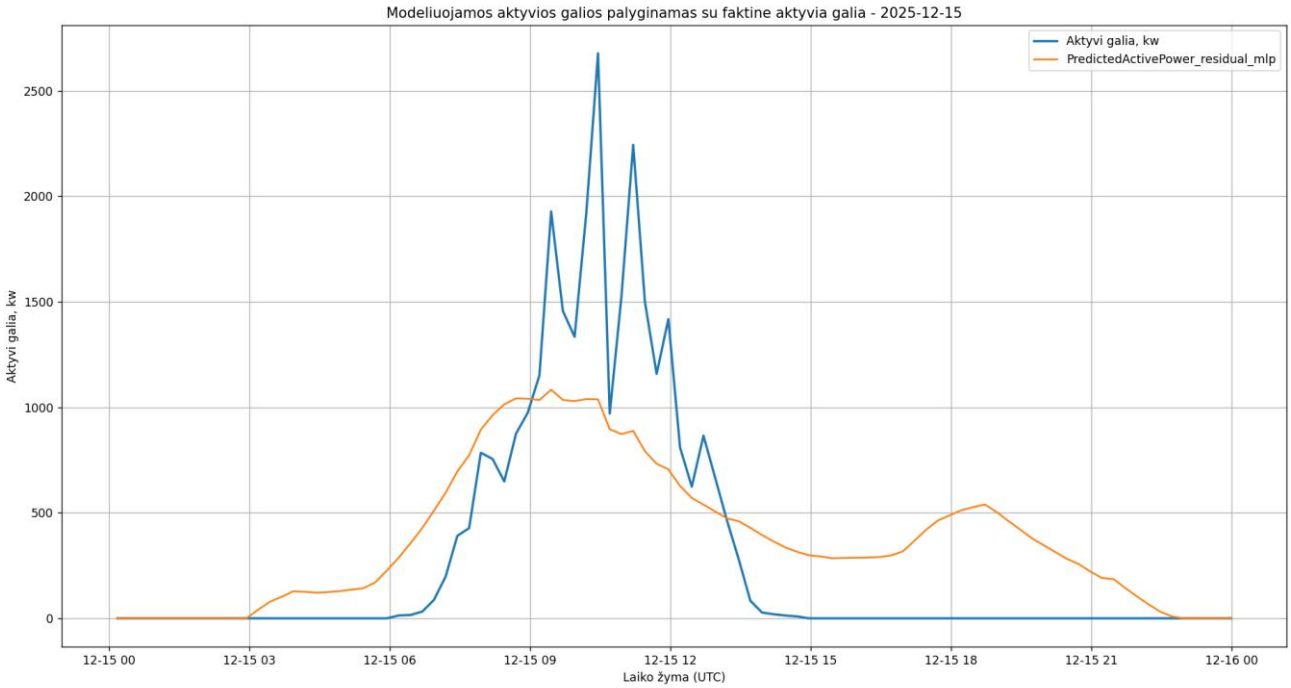
„Json_ArrayDetected“ metodas:

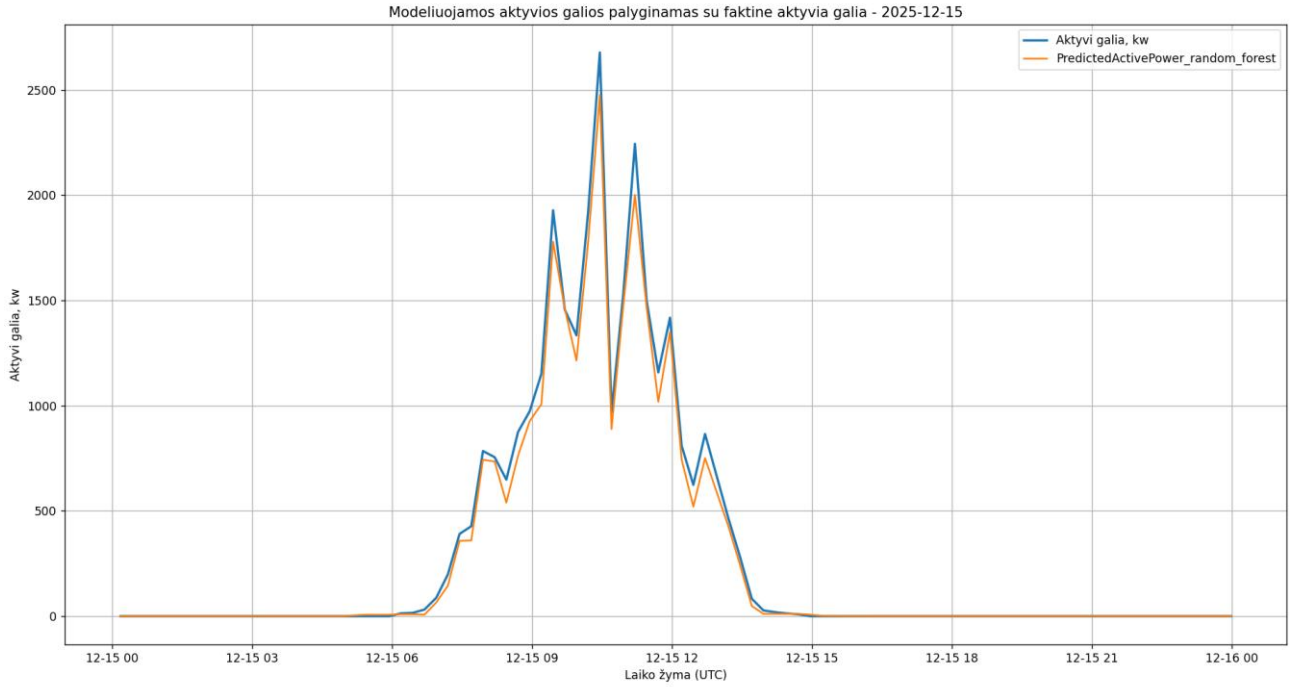
```
(* Aptikus JSON masyvo pradžią *)
ArrayStart := TRUE;
ArrayValue_nr := 1;
```

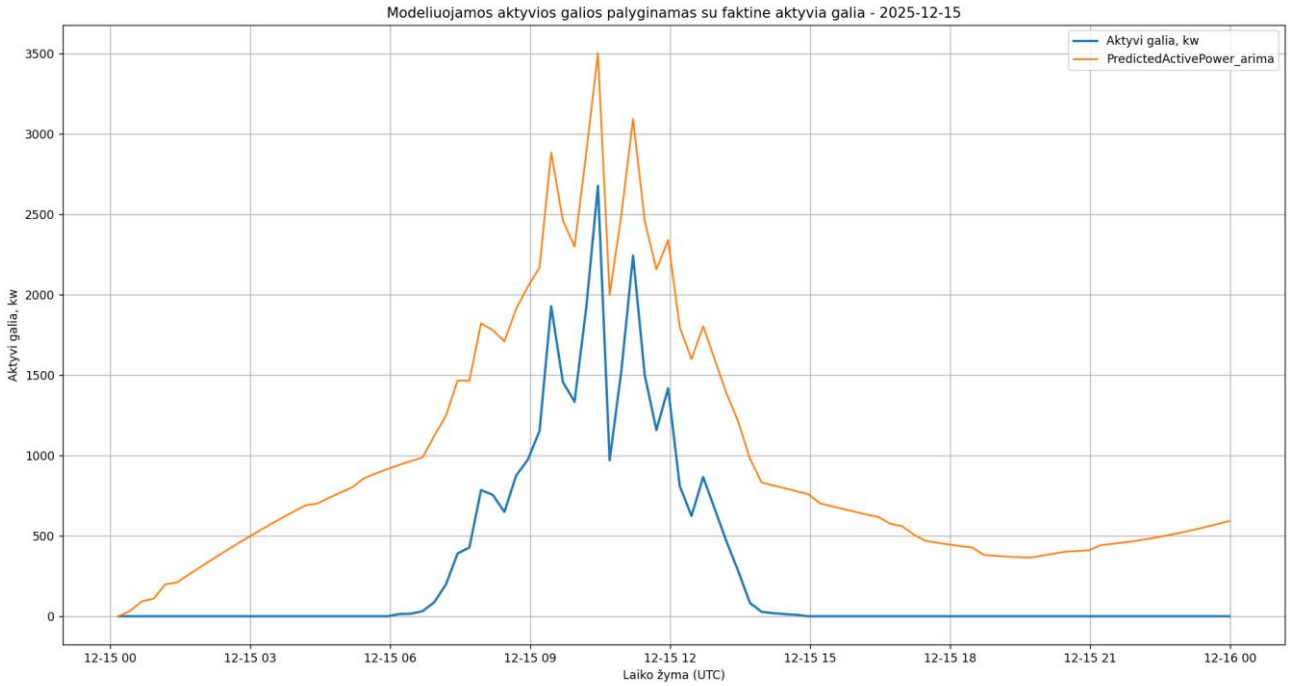
4 priedas. Skirtingų modelių 2025-12-15 grafinis palyginimas su realia tos dienos aktyvia galia (MLP, RF, LSTM, ARIMA)

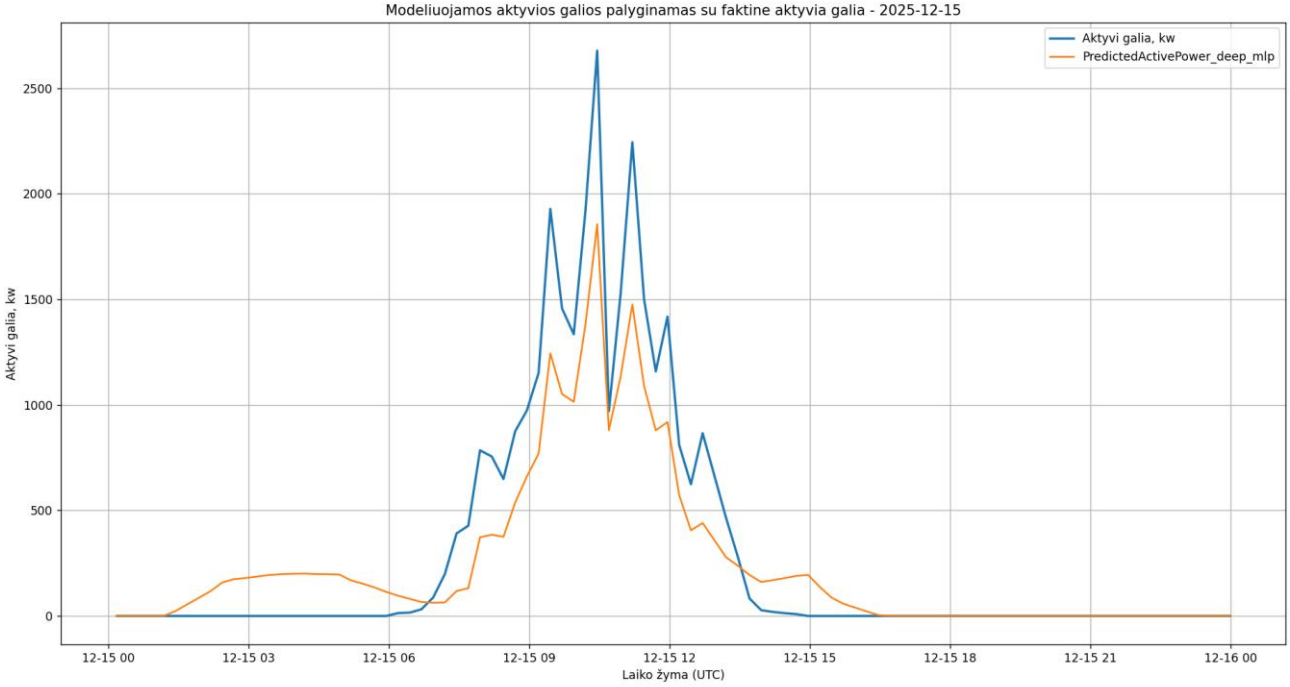

Modeliuojamos aktyvios galios palyginamas su faktine aktyvia galia - 2025-12-15
Aktyvi galia, kw
PredictedActivePower_lstm
Aktyvi galia, kw
Laiko žyma (UTC)
0
500
1000
1500
2000
2500
12-15 00
12-15 03
12-15 06
12-15 09
12-15 12
12-15 15
12-15 18
12-15 21
12-16 00


Modeliuojamos aktyvios galios palyginamas su faktine aktyvia galia - 2025-12-15
Aktyvi galia, kw
PredictedActivePower_mlp
Aktyvi galia, kw
Laiko žyma (UTC)
0
500
1000
1500
2000
2500
12-15 00
12-15 03
12-15 06
12-15 09
12-15 12
12-15 15
12-15 18
12-15 21
12-16 00


Modeliuojamos aktyvios galios palyginamas su faktine aktyvia galia - 2025-12-15
Aktyvi galia, kw
PredictedActivePower_residual_mlp
Aktyvi galia, kw
Laiko žyma (UTC)


Modeliuojamos aktyvios galios palyginamas su faktine aktyvia galia - 2025-12-15
Aktyvi galia, kw
PredictedActivePower_random_forest
Aktyvi galia, kw
Laiko žyma (UTC)


Modeliuojamos aktyvios galios palyginamas su faktine aktyvia galia - 2025-12-15
Aktyvi galia, kw
PredictedActivePower_arima
Aktyvi galia, kw
Laiko žyma (UTC)
0
500
1000
1500
2000
2500
3000
3500
12-15 00
12-15 03
12-15 06
12-15 09
12-15 12
12-15 15
12-15 18
12-15 21
12-16 00


Modeliuojamos aktyvios galios palyginamas su faktine aktyvia galia - 2025-12-15
Aktyvi galia, kw
PredictedActivePower_deep_mlp
Aktyvi galia, kw
Laiko žyma (UTC)
0
500
1000
1500
2000
2500
12-15 00
12-15 03
12-15 06
12-15 09
12-15 12
12-15 15
12-15 18
12-15 21
12-16 00