

ДОДАТОК А

Лістинг коду ANFIS-моделі

```
"""  
  
    ANFIS для прогнозування погодинного споживання  
    електроенергії в ОЕС України.  
  
    Використовує реальну базову криву з ENTSO-E  
  
    """  
  
import numpy as np  
  
import datetime  
  
# Резервна крива (якщо ENTSO-E недоступний)  
  
BASE_LOAD_FALLBACK = [  
  
    11.8, 11.2, 10.8, 10.6, 10.7, 11.0,  
  
    12.2, 13.8, 15.1, 15.7, 15.9, 15.8,  
  
    15.6, 15.4, 15.2, 15.1, 15.4, 16.2,  
  
    16.8, 16.5, 15.7, 14.5, 13.4, 12.5,  
  
    ]  
  
  
# Кешована крива (оновлюється при старті і раз на добу)  
  
_cached_base_load: list[float] | None = None  
  
_cache_updated_at: datetime.datetime | None = None
```

```
BASE_LOAD = BASE_LOAD_FALLBACK # використовується в
роутерах
```

```
def get_base_load_curve() -> list[float]:
```

```
    """Повертає актуальну базову криву навантаження."""
```

```
    if _cached_base_load is not None:
```

```
        return _cached_base_load
```

```
    return BASE_LOAD_FALLBACK
```

```
def set_base_load_curve(curve: list[float]) -> None:
```

```
    """Встановлює криву отриману з ENTSO-E."""
```

```
    global _cached_base_load, _cache_updated_at, BASE_LOAD
```

```
    _cached_base_load = curve
```

```
    _cache_updated_at = datetime.datetime.utcnow()
```

```
    BASE_LOAD = curve
```

```
def seasonal_factor(month: int) -> float:
```

```
    return 1.0 + 0.22 * np.cos(2 * np.pi * (month - 1) /
```

```
12)
```

```
def weekend_factor(day_of_week: int) -> float:
```

```
if day_of_week in (5, 6):

    return 0.88

if day_of_week == 4:

    return 0.97

return 1.0


def temperature_effect(temp_celsius: float) -> float:

    deviation = temp_celsius - 18.0

    if deviation < 0:

        return 1.0 + abs(deviation) * 0.018

    return 1.0 + deviation * 0.008


def cloud_effect(cloud_cover_pct: float) -> float:

    return 1.0 + (cloud_cover_pct / 100.0) * 0.02


def wind_effect(wind_speed_ms: float) -> float:

    return 1.0 - min(wind_speed_ms / 100.0, 0.05)


def anfis_predict(

    hour: int,

    month: int,
```

```

    day_of_week: int,

    temperature: float = 10.0,

    cloud_cover: float = 50.0,

    wind_speed: float = 5.0,

    is_holiday: bool = False,

    is_pre_holiday: bool = False,

    is_school_break: bool = False,

) -> dict:

    """Прогноз на одну годину з використанням реальної
    базової кривої."""

    curve = get_base_load_curve()

    base = curve[hour % 24]

    value = (

        base

        * seasonal_factor(month)

        * weekend_factor(day_of_week)

        * temperature_effect(temperature)

        * cloud_effect(cloud_cover)

        * wind_effect(wind_speed)

    )

```

```

if is_holiday:

    value *= 0.83

elif is_pre_holiday:

    value *= 0.96

if is_school_break:

    value *= 0.94


noise_seed = (hour * 7 + month * 31 + day_of_week * 13)
% 100

noise = (noise_seed / 100.0 - 0.5) * 0.15

value += noise

value = round(max(8.0, min(22.0, value)), 3)

ci_half = round(value * 0.04, 3)


return {

    "forecast": value,

    "lower_bound": round(value - ci_half, 3),

    "upper_bound": round(value + ci_half, 3),

}

```

```

def generate_forecast_series(

    start_iso: str,

    hours: int,

    weather_params: dict,

    calendar_params: dict,

) -> list[dict]:

    """Часовий ряд прогнозів на N годин."""

    start =
datetime.datetime.fromisoformat(start_iso.replace("Z",
"+00:00"))

    results = []

    for i in range(hours):

        ts = start + datetime.timedelta(hours=i)

        ci_multiplier = 1.0 + (i // 6) * 0.01

        pred = anfis_predict(

            hour=ts.hour,

            month=ts.month,

            day_of_week=ts.weekday(),

            temperature=weather_params.get("temperature",
10.0),

```

```

        cloud_cover=weather_params.get("cloud_cover",
50.0),

        wind_speed=weather_params.get("wind_speed",
5.0),

        is_holiday=calendar_params.get("is_holiday",
False),

        is_pre_holiday=calendar_params.get("is_pre_ho
liday", False),

        is_school_break=calendar_params.get("is_schoo
l_break", False),

    )

    ci_half = (pred["forecast"] - pred["lower_bound"])
* ci_multiplier

    results.append({

        "timestamp": ts.isoformat(),

        "hour": ts.hour,

        "forecast": pred["forecast"],

        "lower_bound": round(pred["forecast"] - ci_half,
3),

        "upper_bound": round(pred["forecast"] + ci_half,
3),

        "ci_level": 0.95,

    })

```

```

return results

def compute_metrics(actuals: list[float], predictions:
list[float]) -> dict:

    if not actuals or len(actuals) != len(predictions):

        return {"mape": None, "rmse": None, "mae": None}

    a = np.array(actuals)

    p = np.array(predictions)

    mape = float(np.mean(np.abs((a - p) / a)) * 100)

    rmse = float(np.sqrt(np.mean((a - p) ** 2)) * 1000)

    mae = float(np.mean(np.abs(a - p)) * 1000)

    return {

        "mape": round(mape, 2),

        "rmse": round(rmse, 0),

        "mae": round(mae, 0),

    }

```

Повний вихідний код проекту опубліковано у відкритому репозиторії GitHub за адресою <https://github.com/HerrBonderenko/energy-forecast-ua>

ДОДАТОК Б

Лістинг коду API endpoint формування прогнозу

```

from fastapi import APIRouter, HTTPException
from pydantic import BaseModel, Field
from typing import Optional
import datetime

from app.models import generate_forecast_series, BASE_LOAD

router = APIRouter()

class WeatherParams(BaseModel):
    temperature: float = Field(10.0,
description="Температура (°C)")
    cloud_cover: float = Field(50.0, description="Хмарність (%)")
    wind_speed: float = Field(5.0, description="Швидкість вітру (м/с)")
    humidity: Optional[float] = Field(None,
description="Вологість (%)")
    pressure: Optional[float] = Field(None,
description="Тиск (гПа)")

class CalendarParams(BaseModel):
    is_weekend: bool = False
    is_holiday: bool = False
    is_pre_holiday: bool = False
    is_school_break: bool = False

class ForecastRequest(BaseModel):
    start: str = Field(..., description="ISO 8601 дата початку прогнозу")
    hours: int = Field(24, ge=1, le=168,
description="Горизонт (1-168 годин)")

```

```

weather: WeatherParams = WeatherParams()
calendar: CalendarParams = CalendarParams()

class ForecastPoint(BaseModel):
    timestamp: str
    hour: int
    forecast: float
    lower_bound: float
    upper_bound: float
    ci_level: float

class ForecastResponse(BaseModel):
    request: ForecastRequest
    points: list[ForecastPoint]
    summary: dict
    model_version: str = "v3.1.0"

@router.post("/", response_model=ForecastResponse)
async def create_forecast(req: ForecastRequest):
    """Створити прогноз споживання."""
    try:
        points = generate_forecast_series(
            start_iso=req.start,
            hours=req.hours,
            weather_params=req.weather.model_dump(),
            calendar_params=req.calendar.model_dump(),
        )

        if not points:
            raise HTTPException(status_code=500, detail="Не
вдалося згенерувати прогноз")

        values = [p["forecast"] for p in points]
        max_v = max(values)
        min_v = min(values)
        avg_v = sum(values) / len(values)

```

```

        summary = {
            "avg": round(avg_v, 2),
            "max": round(max_v, 2),
            "min": round(min_v, 2),
            "max_hour": values.index(max_v),
            "min_hour": values.index(min_v),
        }

        return ForecastResponse(request=req, points=points,
                                summary=summary)
    except Exception as e:
        raise HTTPException(status_code=500, detail=str(e))

@router.get("/base-load")
def get_base_load_curve():
    """Базова крива навантаження ОЕС України (24 години)."""
    return {
        "description": "Типовий добовий профіль споживання,  
робочий день",
        "unit": "GW",
        "curve": BASE_LOAD,
    }

@router.get("/preview")
def quick_preview(hours: int = 24):
    """Швидкий прогноз з дефолтними параметрами на N  
годин."""
    now = datetime.datetime.utcnow().replace(minute=0,
                                                second=0, microsecond=0)
    points = generate_forecast_series(
        start_iso=now.isoformat() + "Z",
        hours=hours,
        weather_params={"temperature": 10.0, "cloud_cover":
                        50.0, "wind_speed": 5.0},

```

```

        calendar_params={"is_holiday": False,
        "is_pre_holiday": False, "is_school_break": False},
    )
    return {"start": now.isoformat() + "Z", "points":
points}

@router.get("/actual-load")
async def get_actual_load(hours: int = 24):
    """Реальне споживання з ENTSO-E за останні N годин."""
    from app.services.entsoe_client import
fetch_actual_load, is_configured
    import datetime

    if not is_configured():
        return {"status": "not_configured", "data": []}

    now = datetime.datetime.utcnow().replace(minute=0,
second=0, microsecond=0)
    start = now - datetime.timedelta(hours=hours)

    data = await fetch_actual_load(start, now)
    return {
        "status": "ok",
        "source": "ENTSO-E",
        "country": "Ukraine (UA_IPS)",
        "points": len(data),
        "data": data,
    }

@router.get("/debug-entsoe")
async def debug_entsoe():
    """Debug - діагностика ENTSO-E підключення."""
    import os
    import datetime
    import aiohttp
    from app.services.entsoe_client import (

```

```

ENTSOE_URL,    UKRAINE_EIC,    _get_token,
fetch_actual_load
    )

token = _get_token()
if not token:
    return {"error": "no token"}

# Спробуємо різні EIC коди для України
eic_codes = [
    ("UA_IPS",    "10Y1001C--00003F"),
    ("UA-BEI",    "10YUA-WEPS-----0"),
    ("UA",        "10YUA-----A"),
]

now = datetime.datetime.utcnow().replace(minute=0,
second=0, microsecond=0)
start = now - datetime.timedelta(days=2)

results = {}
for name, eic in eic_codes:
    params = {
        "securityToken": token,
        "documentType": "A65",
        "processType": "A16",
        "outBiddingZone_Domain": eic,
        "periodStart": start.strftime("%Y%m%d%H%M"),
        "periodEnd":    now.strftime("%Y%m%d%H%M"),
    }
    try:
        async with aiohttp.ClientSession() as session:
            async with session.get(
                ENTSOE_URL, params=params,
                timeout=aiohttp.ClientTimeout(total=20)
            ) as resp:
                text = await resp.text()

```

```
        results[name] = {
            "eic": eic,
            "status": resp.status,
            "response_preview": text[:500],
        }
    except Exception as e:
        results[name] = {"eic": eic, "error": str(e)}

return {
    "token_length": len(token),
    "period_start": start.strftime("%Y%m%d%H%M"),
    "period_end": now.strftime("%Y%m%d%H%M"),
    "tests": results,
}
```

ДОДАТОК В

Лістинг коду клієнта API

```

const BASE_URL = import.meta.env.VITE_API_URL ||
'http://localhost:8000';

async function request(path, options = {}) {
  const url = `${BASE_URL}${path}`;
  try {
    const res = await fetch(url, {
      headers: { 'Content-Type': 'application/json' },
      ...options,
    });
    if (!res.ok) {
      const err = await res.text();
      throw new Error(`API ${res.status}: ${err}`);
    }
    return await res.json();
  } catch (e) {
    console.error(`[API] ${path}:`, e.message);
    throw e;
  }
}

// ————— Погода —————
export async function getCurrentWeather() {
  return request('/api/weather/current');
}

export async function getWeatherForecast(hours = 24) {
  return request(`/api/weather/forecast?hours=${hours}`);
}

// ————— Прогноз —————

```

```

export async function createForecast({ start, hours,
weather, calendar }) {
    return request('/api/forecast/', {
        method: 'POST',
        body: JSON.stringify({ start, hours, weather, calendar
    })),
    });
}

export async function getForecastPreview(hours = 24) {
    return request(`/api/forecast/preview?hours=${hours}`);
}

export async function getBaseLoadCurve() {
    return request('/api/forecast/base-load');
}

// — Модель —————
export async function getModelInfo() {
    return request('/api/model/info');
}

export async function getModelMetrics() {
    return request('/api/model/metrics');
}

export async function getFuzzyRules() {
    return request('/api/model/rules');
}

// — Історія —————
export async function getHistory({ days = 7, limit = 50 }
= {})) {
    return
    request(`/api/history/?days=${days}&limit=${limit}`);
}

```

```
// — Health check —————  
export async function checkHealth() {  
  try {  
    const data = await request('/health');  
    return data.status === 'ok';  
  } catch {  
    return false;  
  }  
}
```

ДОДАТОК Г

Відомість кваліфікаційної роботи

Позначення					Найменування		Дод. відомості	
					Текстові документи			
1.					Пояснювальна записка		90 с.	