

Python and AI/ML for Weather, Climate and Environmental Applications



Let us enjoy 🚀
playing 🤖 with
Python 🐍 and AI/ML!
 

Five-Day Schedule Overview

Time	Day 1	Day 2	Day 3	Day 4	Day 5
09:00–10:00	Opening by ECMWF DG, Start: Coding & Science in the Age of AI	Neural Network Architectures	Diffusion and Graph Networks	MLOps Foundations	Model Emulation, AIFS and AICON
10:00–11:00	Lab: Python Startup: Basics	Lab: Feed-forward and Graph NNs	Lab: Graph Learning with PyTorch	Lab: Containers and Reproducibility	Lab: Emulation Case Studies
11:00–12:00	Python, Jupyter and APIs	Large Language Models	Agents and Coding with LLMs	CI/CD for Machine Learning	AI-based Data Assimilation
12:00–12:45	Lab: Work environments, Python everywhere	Lab: Simple Transformer and LLM Use	Lab: Agent Frameworks	Lab: CI/CD Pipelines	Lab: Graph-based Assimilation
12:45–13:30	Lunch Break				
13:30–14:30	Visualising Fields and Observations	Retrieval-Augmented Generation (RAG)	DAWID System and Feature Detection	Anemoi: AI-based Weather Modelling	AI and Physics
14:30–15:30	Lab: GRIB, NetCDF and Obs Visualisation	Lab: RAG Pipeline	Lab: DAWID Exploration	Lab: Anemoi Training Pipeline	Lab: Physics-informed Neural Networks
15:30–16:15	Introduction to AI and Machine Learning	Multimodal Large Language Models	MLflow: Managing Experiments	The AI Transformation	Learning from Observations Only
16:15–17:00	Lab: Torch Tensors and First Neural Net	Lab: Radar, SAT and Multimodal Data	Lab: MLflow Hands-on	Lab: How work style could change	Lab: ORIGEN and Open Discussion
17:00–20:00					Joint Dinner

DAWID in Action

User: rpotthast | Logout New Session 

Session ID: 63e08dff07f356340512ac4b94f39b8 Sessions

DAWID beta

You asked:

what is the weather tomorrow at 3pm in Frankfurt?

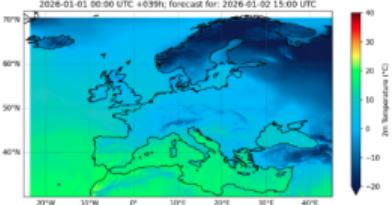
DAWID:

- Model
- Function get_weather_forecast Results
- ▼ Model

gpt-5-mml

Here is the ICON-EU forecast plot for Frankfurt tomorrow (2026-01-02) at 15:00 (03:00 PM) — see the image below:

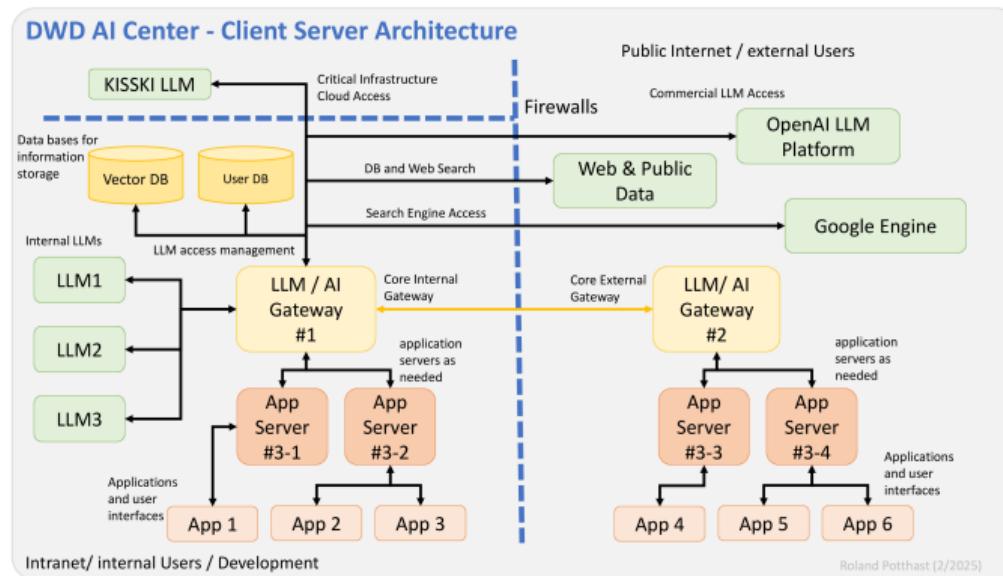
ICON-EU 2m Temperature (°C)
2026-01-01 00:00 UTC +059h; forecast for: 2026-01-02 15:00 UTC



Would you like a numeric summary (temperature, precipitation chance, wind speed/direction) or a short text forecast? I can extract those values for you.

Interactive DAWID assistant session with live streaming responses and tool integration

AI Centre Client–Server Architecture



Frontend UI, backend orchestration, language models, tools, and data sources

DAWID System Architecture — Frontend Stack

User Interface Layer

- ▶ **HTML5** Page structure, forms, UI elements
- ▶ **CSS** Layout, styling, responsive design
- ▶ **JavaScript** Interaction logic and event handling
- ▶ **Fetch API** Asynchronous requests and streaming

Dynamic Content

- ▶ **Markdown rendering** Incremental formatting of streamed output

Browser Capabilities

- ▶ **Audio** recording via Web APIs
- ▶ Clipboard access
- ▶ Local UI state and session handling

Design Principle

- ▶ No heavy frontend frameworks
- ▶ No client-side AI logic
- ▶ Frontend as thin interaction layer
- ▶ Works on **Laptop**, Workstation, **Mobile Device**

DAWID System Architecture — Server and Backend

Web and Gateway Layer

- ▶ **Web Server** (Apache / Nginx)
TLS termination, static content, routing
- ▶ **PHP Gateway** Sessions, uploads, request forwarding
- ▶ **Security Boundary** No direct access to backend services

Role

- ▶ Separates public UI from compute backend
- ▶ Enforces access control and isolation

Backend and Compute Layer

- ▶ **Python Backend (FastAPI)**
Central orchestration logic
- ▶ **Uvicorn Server** Async execution, streaming responses
- ▶ **LLM Backends** Local and cloud-based models

Key Principle

- ▶ Backend owns state, context, and tools
- ▶ Frontend remains stateless and simple

File Upload Workflow in the DAWID Frontend

User Interaction

- ▶ File selected via HTML form
- ▶ Automatic submission on selection
- ▶ No explicit upload button required

Gateway Interaction

- ▶ Upload sent to PHP gateway
- ▶ File stored in session-specific folder
- ▶ Backend notified of new resource

Frontend Logic

- ▶ JavaScript listens to file change events
- ▶ Uses **Fetch API** for upload
- ▶ Asynchronous, non-blocking UI

Minimal JavaScript Logic

Automatic upload on file selection

```
1 fileInput.addEventListener("  
  change", () => {  
  2   uploadForm.requestSubmit();  
  3 });
```

Streaming LLM Responses in the Browser

Why Streaming?

- ▶ Immediate user feedback
- ▶ Reduced perceived latency
- ▶ Long answers remain usable

Frontend Mechanism

- ▶ **Fetch API** with streamed response
- ▶ **ReadableStream** reader
- ▶ Chunk-by-chunk text processing

Key UX Effect

- ▶ User sees **partial results** instantly
- ▶ No blocking on full completion

Incremental Rendering

- ▶ Accumulate streamed text
- ▶ Re-render on each new chunk
- ▶ Markdown parsed continuously

```
const reader = response.body.getReader();
while (true) {
    const { done, value } = await reader.read();
    if (done) break;
    partial += decoder.decode(value);
    responseDiv.innerHTML = marked.parse(partial)
}
```

Design Choice

- ▶ Simple, robust rendering

Frontend Capabilities: Math Rendering and Sessions

Streaming Mathematical Content

1. Proof by similar triangles

Let triangle ABC be right-angled at C . Put $AB = c$ (hypotenuse), $BC = a$, $AC = b$ the foot be D . Let $AD = d$, $DB = e$ so $d + e = c$. From similarity of triangles:

$$\frac{a}{c} = \frac{c}{e} \Rightarrow a^2 = c \cdot e$$

$$\frac{b}{c} = \frac{c}{d} \Rightarrow b^2 = c \cdot d$$

Adding:

$$a^2 + b^2 = c(e + d) = c \cdot c = c^2.$$

2. Area / rearrangement proof (square with four triangles)

Mathematical expressions rendered live during streamed responses using **Markdown** and **MathJax**.

Session Display and Management



Session Name	Last Access	ID
Frankfurt Forecast & Pythagoras Proof...	1.1.2026, 11:12:21	63e08dff077356340512a4b94f39b8
Small talk with DAWID	22.12.2025, 21:38:01	e969085878b2c9fffc24803508befa2e
new session	16.12.2025, 16:50:04	e26bf7923925e5b7ae371526dc870c50
new session	16.12.2025, 16:44:27	b1148044d2c9264362d37822453620d8
Frankfurt — Tomorrow 9:00 Forecast	16.12.2025, 16:43:19	

Multiple interaction sessions with selectable history and clear **context separation** at the frontend level.

DAWID Backend: Central Unicorn Server

Role of the Backend Server

- ▶ Single central entry point for all requests
- ▶ Receives queries from the frontend gateway
- ▶ Handles sessions and user context
- ▶ Streams responses back to the client

Key Responsibilities

- ▶ Model selection and routing
- ▶ Context assembly (history, RAG)
- ▶ Tool and workflow execution

FastAPI + Unicorn

- ▶ **FastAPI**: API definition and validation
- ▶ **Unicorn**: async, high-performance server
- ▶ Multi-worker capable deployment

Design Principle

- ▶ Backend owns state, logic, and control
- ▶ LLMs are subsystems, not the system

Backend Route Overview

Main Route Categories

- ▶ **LLM Routes** Prompt handling, streaming responses
- ▶ **Upload Routes** File transfer, storage, metadata
- ▶ **Audio Routes** Speech upload and transcription

Supporting Routes

- ▶ Dataspace and document access
- ▶ Session and user management

API Routes

- ▶ OpenAI-compatible chat API
- ▶ Streaming and non-streaming endpoints
- ▶ Programmatic access for external tools

Design Principles

- ▶ Clear separation of concerns
- ▶ Explicit routing and ownership
- ▶ Routes reflect system capabilities

LLM Streaming Routes

Purpose of LLM Routes

- ▶ Accept user queries from the gateway
- ▶ Build prompts from session history
- ▶ Integrate retrieved context (RAG)
- ▶ Return responses as streams

Streaming Logic

- ▶ Token-wise or chunk-wise output
- ▶ Immediate forwarding to frontend
- ▶ Non-blocking async execution

Unified Interface

- ▶ Same route for local and cloud models
- ▶ OpenAI, Claude, Gemini, LLaMA, etc.
- ▶ Model chosen at runtime

Interception Points

- ▶ Function call detection
- ▶ Tool execution triggers
- ▶ Session state updates

Key Insight

- ▶ LLM output is **not final**
- ▶ It is an intermediate signal

Upload and File Management Routes

Purpose of Upload Routes

- ▶ Receive files from the frontend gateway
- ▶ Store data in session- or user-specific folders
- ▶ Register files for later access by tools

File Types

- ▶ Documents (PDF, text, Markdown)
- ▶ Data files (NetCDF, images)
- ▶ Audio recordings

Backend Handling

- ▶ Controlled directory structure
- ▶ Metadata stored with session context
- ▶ Optional indexing for retrieval

Integration Points

- ▶ Retrieval-Augmented Generation (RAG)
- ▶ Tool execution (plots, analysis)
- ▶ Download and reuse in workflows

Key Principle

- ▶ Uploaded files become first-class resources

Audio Routes: Speech-to-Text

Motivation

- ▶ Natural user interaction via speech
- ▶ Hands-free input for complex queries
- ▶ Multimodal access without UI complexity

Frontend Interaction

- ▶ Audio recorded in the browser
- ▶ File uploaded via gateway
- ▶ Treated like any other uploaded resource

Backend Processing

- ▶ Local or remote speech-to-text engine
- ▶ Conversion to plain text
- ▶ Injected into standard LLM pipeline

Design Principle

- ▶ Audio is a preprocessing step
- ▶ All downstream logic remains text-based
- ▶ No special cases for speech afterwards

User Management and Dataspaces

User and Session Model

- ▶ Each interaction tied to a session ID
- ▶ Optional persistent user identity
- ▶ Separation of short-term and long-term context

Access Control

- ▶ User-specific private areas
- ▶ Controlled access to shared resources

Dataspaces

- ▶ Private document folders per user
- ▶ Group-shared knowledge spaces
- ▶ Clear ownership and responsibility

Purpose

- ▶ Organize documents and data
- ▶ Provide structured context to the LLM
- ▶ Enable collaborative workflows

Retrieval-Augmented Generation (RAG)

Motivation

- ▶ LLMs do not know user-specific documents
- ▶ Uploaded data must influence responses
- ▶ Context must be selected , not dumped

Local Retrieval

- ▶ Document chunks embedded locally
- ▶ Similarity search via **FAISS**
- ▶ No external data transfer

Context Assembly

- ▶ Detect when documents are relevant
- ▶ Retrieve top-matching chunks
- ▶ Inject into prompt before generation

Key Principle

- ▶ RAG augments knowledge , not reasoning
- ▶ The LLM remains the decision engine

Available LLM Models and Capability Tiers

Capability Tiers

- ▶ **FAST** Lightweight, fast, cost-efficient
- ▶ **CORE** Strong general-purpose models
- ▶ **PRO** Heavy reasoning, coding, long context
- ▶ **ULTRA** Highest available model capability

Design Choice

- ▶ One **best model** per tier and supplier
- ▶ No artificial or redundant model options

Supported Model Families

- ▶ OpenAI (Gpt5.2, GPT5.1, GPT-4o, GPT-5-mini)
- ▶ Claude (Anthropic)
- ▶ Gemini (Google)
- ▶ LLaMA (local and remote)
- ▶ Mistral / Mixtral
- ▶ GPT-OSS

Key Principle

- ▶ Users select **capability**, not internals
- ▶ Backend resolves tier to concrete model

DAWID Model Aliases and Routing

Internal Model Aliases

- ▶ Abstract names used by the frontend
- ▶ Examples:
 - ▶ openai-fast, openai-core
 - ▶ claude-pro
 - ▶ llama-fast
- ▶ Aliases encode **capability tier**

Why Aliases?

- ▶ Stable interface for users
- ▶ Models can change transparently

Backend Resolution

- ▶ Alias mapped to concrete model name
- ▶ Mapping defined centrally in settings
- ▶ Supplier-specific resolution

Routing Logic

- ▶ Backend checks model ownership
- ▶ Selects local or remote backend
- ▶ Routes request to correct engine

Key Principle

- ▶ Frontend never sees real model IDs
- ▶ Backend retains **full control**

Function Calling in DAWID: Weather Service Productivity

DAWID as a Productivity Platform

- ▶ DAWID supports internal and external weather services
- ▶ Focus on operational and scientific workflows
- ▶ Integration of data, models, and tools

Role of Function Calling

- ▶ LLM suggests domain-relevant actions
- ▶ DAWID executes approved functions
- ▶ Actions are embedded in workflows

Typical Weather-Service Tasks

- ▶ Access and process NWP data
- ▶ Generate plots, maps, and diagnostics
- ▶ Perform domain-specific analyses
- ▶ Support decision-making and reporting

Key Principle

- ▶ Function calls increase efficiency, not autonomy
- ▶ **DAWID augments expert work, it does not replace it**

Classical Function Calling in DAWID (JSON-Based)

What “Classical” Means in DAWID

- ▶ Function calls encoded explicitly in text
- ▶ Usually formatted as **JSON blocks**
- ▶ Independent of specific LLM providers

Why DAWID Still Supports This

- ▶ Works with **all models** (local and remote)
- ▶ Robust fallback mechanism
- ▶ Easy to debug and inspect

Example: Function Proposal by the LLM

JSON-style function call

```
1 {"function_calls": [{}  
2   "name": "get_icon_forecast  
3   ",  
4   "arguments": {  
5     "variable": "t2m",  
6     "region": "Germany" }]}]
```

DAWID Interpretation

- ▶ JSON is **parsed and validated**
- ▶ Function is checked against allow-list
- ▶ Execution only happens in backend

Native Function Calling in DAWID

What “Native” Means

- ▶ Functions declared via explicit schemas
- ▶ Typed arguments and clear signatures
- ▶ Direct support by modern LLM APIs

Advantages in DAWID

- ▶ No JSON parsing from free text
- ▶ Fewer hallucinated calls
- ▶ Safer execution path

DAWID Integration

- ▶ Functions registered centrally
- ▶ Only allowed tools are exposed
- ▶ Backend controls execution order

Typical Use Cases

- ▶ Plot generation
- ▶ Data download and extraction
- ▶ Model-based diagnostics

Design Principle

- ▶ Native calls are preferred
- ▶ Classical calls remain a fallback

Function Calling inside DAWID: Execution Paths

Internal Execution: LangGraph

- ▶ Functions organized as LangGraph nodes
- ▶ Explicit shared state passed between nodes
- ▶ Deterministic execution order
- ▶ Suitable for multi-step workflows

Typical Internal Tasks

- ▶ Data preprocessing and analysis
- ▶ Plot and image generation
- ▶ Chained scientific workflows

External Execution: HTTP Endpoints

- ▶ Functions exposed via REST endpoints
- ▶ Called from DAWID via HTTP requests
- ▶ External services remain isolated

Typical External Tasks

- ▶ Access to targeted data services
- ▶ Specialized model inference
- ▶ **enables modular service development**

Key Principle

- ▶ function can run on various computers
- ▶ Execution remains controlled and auditable, flexible, cloud ready

External Function Calls inside DAWID

```
# === External Client API forwarding ===
external_funcs = [f for f in functions_requested if f["name"] in EXTERNAL_FUNCTIONS]

print(f"external_funcs={external_funcs}")

if external_funcs:
    fc = external_funcs[0]

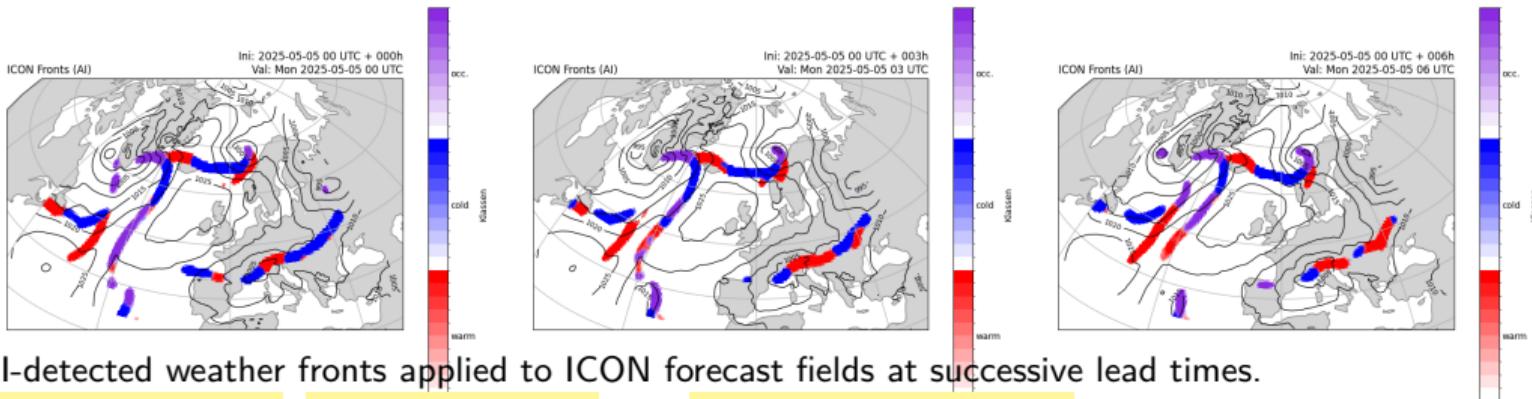
    payload = {
        "session_id": session_id,
        "username": username,
        "function_call": fc
    }

EXTERNAL_URL = os.getenv("EXTERNAL_CLIENT_API", "http://localhost:5000/api/forward")

try:
    import requests
    r = requests.post(EXTERNAL_URL, json=payload, timeout=20)
    r.raise_for_status()
    return f"/ External client forwarded: {r.text}"
except Exception as e:
    return f"✗ External client call failed: {e}"
```

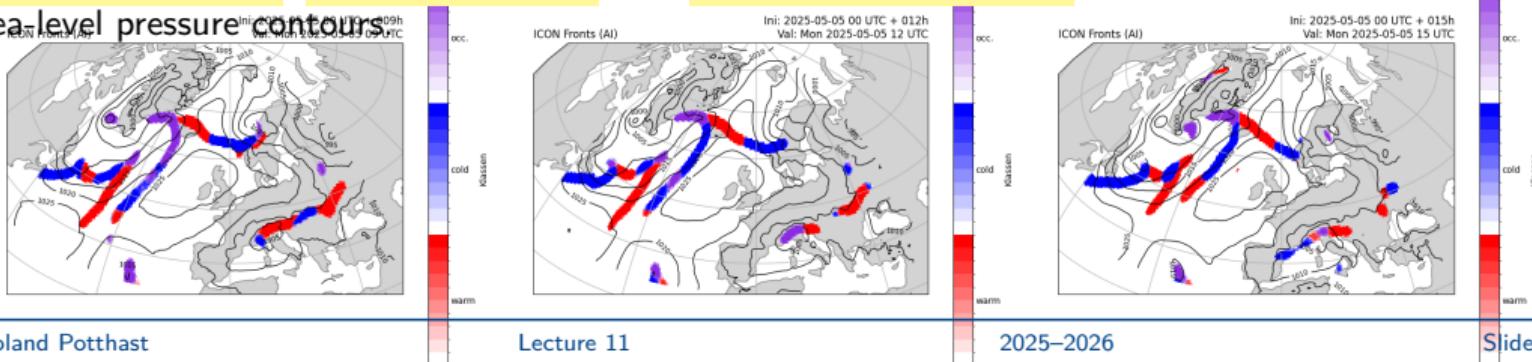
Integration of **external function calls** within the DAWID backend through REQUESTS.

AI-Based Feature Detection: Weather Fronts



AI-detected weather fronts applied to ICON forecast fields at successive lead times.

Warm fronts (red), cold fronts (blue), and occluded fronts (violet) are overlaid with mean sea-level pressure contours.



Input Data and Preprocessing

Meteorological Input Fields

- ▶ Mean sea-level pressure (**PMSL**)
- ▶ 2 m temperature (**T2M**)
- ▶ 2 m relative humidity (**RH2M**)
- ▶ 10 m wind components (**U10M**,
V10M)
- ▶ Land-sea mask (**FRLAND**)

Data Source

- ▶ ICON analysis and forecast fields
- ▶ Regular latitude-longitude grid

Preprocessing Steps

- ▶ Channel-wise normalization
- ▶ Scaling to comparable numerical ranges
- ▶ Binary encoding of land-sea mask

Resulting Tensor

- ▶ Shape: [6, lat, lon]
- ▶ Stored in NetCDF format
- ▶ One file per analysis or forecast time

Design Principle

- ▶ Preserve physical meaning
- ▶ Avoid unnecessary feature engineering

Model Architecture and Training

Model Architecture

- ▶ U-Net for semantic segmentation
- ▶ Encoder-decoder with skip connections
- ▶ Preserves spatial detail

Input / Output

- ▶ Input channels: 6
- ▶ Output classes: 4
- ▶ Pixel-wise classification

PyTorch Model Setup

```
model = UNet(  
    in_channels=6,  
    out_channels=4,  
    init_features=64  
).to(device)  
  
loss_fn = nn.CrossEntropyLoss()  
optimizer = torch.optim.AdamW(  
    model.parameters(),  
    lr=1e-4 )
```

Training Setup

- ▶ Supervised learning
- ▶ Labeled frontal maps
- ▶ Early stopping and validation

Operational Front Detection in Forecast Practice

From Research to Operations

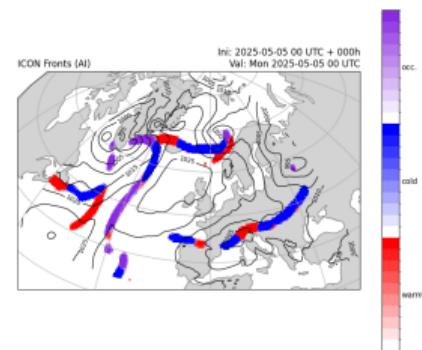
- ▶ AI-based front detection is operationally deployed
- ▶ Integrated into NinJo service stations
- ▶ Available directly in the forecasters' workflow

Operational Value

- ▶ Fast, consistent front analysis
- ▶ Supports situational awareness
- ▶ Reduces manual drawing effort

User Acceptance

- ▶ Widely used by weather forecasters
- ▶ Particularly helpful in complex synoptic situations
- ▶ Acts as decision support, not automation



Lecture 11 — Summary

DAWID Platform

- ▶ **Productivity platform** for NHMS
- ▶ Supports internal and external users
- ▶ Lightweight web frontend
- ▶ Streaming interaction and session management

Backend Architecture

- ▶ Modular FastAPI backend
- ▶ Unified access to multiple LLM families
- ▶ Controlled function execution
- ▶ Local and external services integrated

AI Integration

- ▶ Function calling via LangGraph workflows
- ▶ External tools accessed through validated APIs
- ▶ Retrieval-Augmented Generation RAG using FAISS

Operational Feature Detection

- ▶ AI-based feature detection in operations
- ▶ Front detection integrated into NinJo
- ▶ High acceptance among forecasters