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	

The AI Transformation: Communication History in 8 Steps

From printed knowledge to mass media

- **Book printing** 1450
  (Gutenberg press, scalable knowledge)
- **Electricity / telegraph** 1837
  (Morse telegraph, instant signals)
- **Radio** 1895
  (wireless transmission, mass broadcasting)
- **Television** 1927
  (moving images, global events)

Transformation:
Communication turns into interaction.

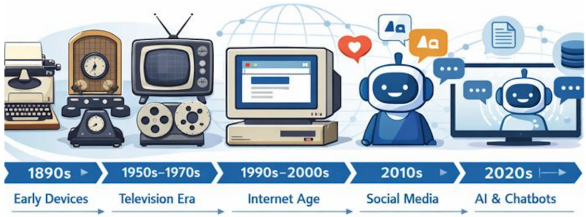
From computation to language interfaces

- **Computers** 1941
  (Konrad Zuse Z3, programmable digital computer)
- **Internet** 1983
  (TCP/IP, global connectivity)
- **WWW** 1993
  (Internet for all!)
- **Social media** 2004
  (Facebook era, interactive networks)
- **LLMs & AI systems** 2022
  (language becomes an interface , interaction at scale)

Why Now? The Next Step Has Arrived

A new communication revolution

- Book printing scaled **knowledge**
- Internet scaled **connectivity**
- **LLMs scale interaction:**
  language becomes an interface



Progress in Communication.
What does it mean?
Where are we as humans?

What changes:

- from **information access**
- to **interactive assistance**

This affects communication, documentation, coding, workflows

— across the entire organisation.

Why Now? Language Becomes an Interface

What changed in 2022?

- For the first time, computers can handle natural language appropriately
- This enables **interaction** instead of programming
- Not only answers — but **actions**:
  (writing, summarizing, translating, coding, planning)

New capability: conversational work

- Ask → draft → revise → finalize
- Explain → implement → test → debug

Examples in daily work

- Draft emails, minutes, reports
- Summarize long documents
- Create slides from bullet points
- Helpdesk: FAQ, ticket triage
- Code assistance: refactor + docs

Key message:

AI affects everyone — not only scientists or IT staff.

Slow Down: 1993: The Internet Becomes Public

1993: a global information layer

- Web browsers make the Internet usable for everyone
- Information access becomes **self-service**
- Knowledge shifts from **local** to **global**

Key change:
Search and access replace distribution.

Many workflows become web-based: documentation, support, collaboration, publishing.

Internet era: global connectivity and instant access

2004 / 2008: Social Media + Smartphones

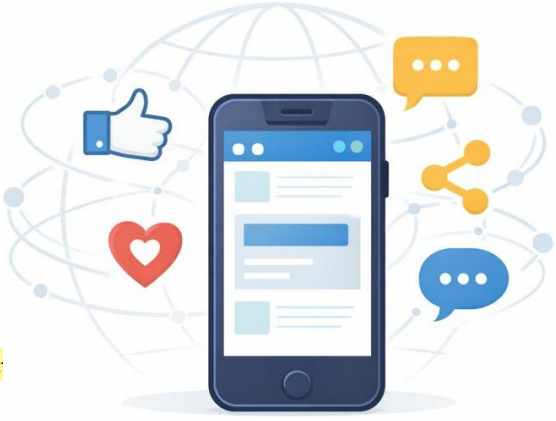
From websites to interactive networks

- 2004: social networks go mainstream
- 2007/2008: smartphones put the Internet in every pocket
- Everyone becomes: **consumer + producer + distributor**

Key change:

Communication becomes continuous and interac

Attention becomes a resource; content flows in real time.



The always-on era: interaction at global scale

2012–2019: The ML Revolution in Vision and Translation

Deep learning becomes practical

- **Image recognition** jumps to new accuracy levels
- **Machine translation** improves dramatically
- Pattern learning outperforms hand-crafted rules

Modern deep learning architectures enable scaling

Key change:
Perception tasks become learnable at scale.

This is the foundation for today's multi-modal AI (text, images, audio).

2022: The Chatbot Threshold

The breakthrough: language interaction

- Chatbots become useful for real work
- People can **ask, refine, and iterate** in dialogue
- The interface is now natural language

Key change:
From communication to interaction.

LLMs + tools + knowledge access ⇒ AI assistants and agents.

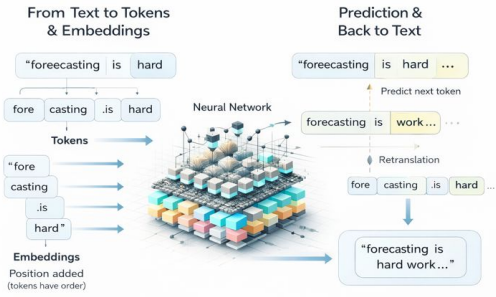
Chatbots evolve into systems: retrieval + tools + guardrails

LLMs in One Sentence

A large language model is a prediction machine

- It reads text and predicts **the next token**
- It repeats this step many times to produce an answer
- Training: learn statistical structure from huge corpora

Key point: It does not store facts like an encyclopedia. It learns patterns of language and reasoning.



Transformer models: sequence in → sequence out

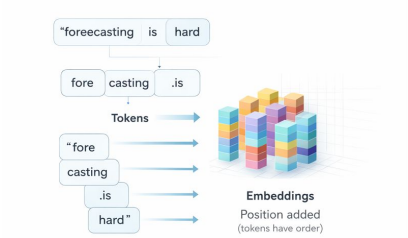
Step 1: From Text to Tokens

Computers do not read words

- Text is split into **tokens**
- Tokens are pieces of words, punctuation, spaces
- Example:

```
"forecasting is hard"
⇒ ["fore", "casting", " is", " hard"]
⇒ [1523, 9182, 318, 6732]   (token IDs)
```

Why tokens matter: they define what the model can represent efficiently.



Tokens → vectors

- each token becomes a vector (**embedding**)
- vectors capture similarity:
  ("rain" closer to "cloud" than to "banana")
- position is added:
  (word order matters)

This is how text enters a neural network: as numbers in a high-dimensional space.

Step 2: Attention — How the Model Understands Context

Attention is selective reading

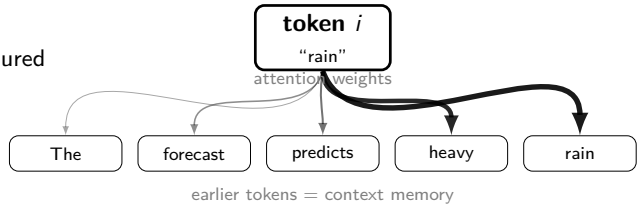
- for each word, the model decides: **what earlier words matter**
- it forms weighted links between tokens
- repeated many times in layers

Key point: Attention allows long-range dependencies and structured reasoning.

A simple intuition

- Like humans: **scan** + **focus** + **connect**
- Not one focus, but many:
  (**multi-head attention**)

Transformer = many attention layers + feed-forward layers.



Step 3: Training — Next Token Prediction

Training objective

- show text to the model
- hide the next token
- train it to predict the hidden token

Why it can fail

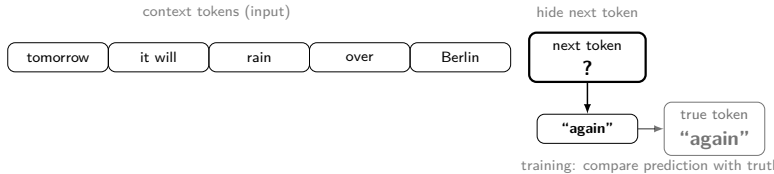
- it predicts **plausible text**
- not guaranteed truth
- can **hallucinate** details

Result: the model learns grammar, style, facts, and reasoning patterns as an emergent capability.

Therefore: verification + grounding are crucial in professional use.

Important: training is expensive (compute + data).

Using a model is cheap (inference).



Inference: Prompting Is Steering

A prompt creates the context

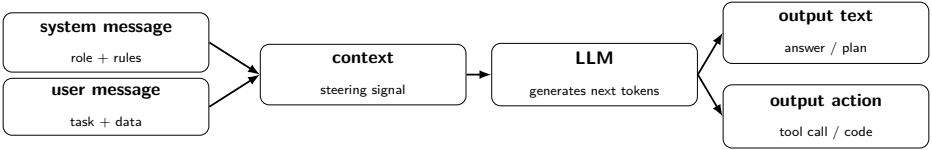
- system message: role + rules
- user message: task + data
- the model continues from there

Key point: The model is not stable by itself. It is shaped by the context we provide.

What improves results

- clear goal + constraints
- examples of desired style
- provide trusted information
- ask for uncertainties / checks

Good prompting is a communication skill: precise intent, good context, clear quality criteria.



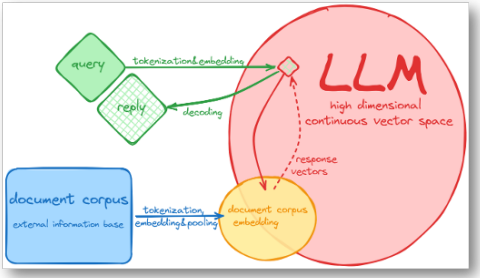
Prompting = setting context that steers the model's behavior and output.

From Language to Interaction: LLMs + Tools

LLMs become useful at work when connected

- **retrieval (RAG):** trusted documents
  - policies, manuals, guidance, internal knowledge bases
  - reduces hallucinations by grounding on sources
- **tools:** APIs, code execution, search
  - from "talking" to doing (compute, query, automate)
  - reproducible outputs: scripts, plots, tables
- **memory:** project and user context
  - remembers assumptions, preferences, ongoing tasks
  - avoids repeating the same onboarding every time

Key message: Real systems are LLM-based assistants, not just chat.



From Chat to Action: Why Function Calling Matters

Chat is not the goal.

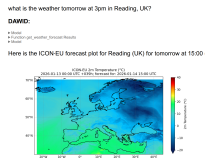
LLMs become truly useful when they can trigger real actions:

- download and inspect forecast data
- compute diagnostics and key numbers
- generate plots and reports
- call services (archives, NWP, HPC tools)

Function calling turns language into structured decisions — executed by a controlled backend (not by the LLM).

Example: tool call proposed by the LLM

```
{
  "tool_name": "get_weather_forecast",
  "arguments": {
    "model": "ICON-EU",
    "variable": "t2m",
    "location": "Berlin",
    "lead_time_h": 24
  }
}
```



DAWID executes the tool call and returns the result.

Tool Awareness: How the Model Knows What It Can Do

The key idea

The LLM cannot "invent" actions. It can only call tools that we explicitly provide.

Why this matters

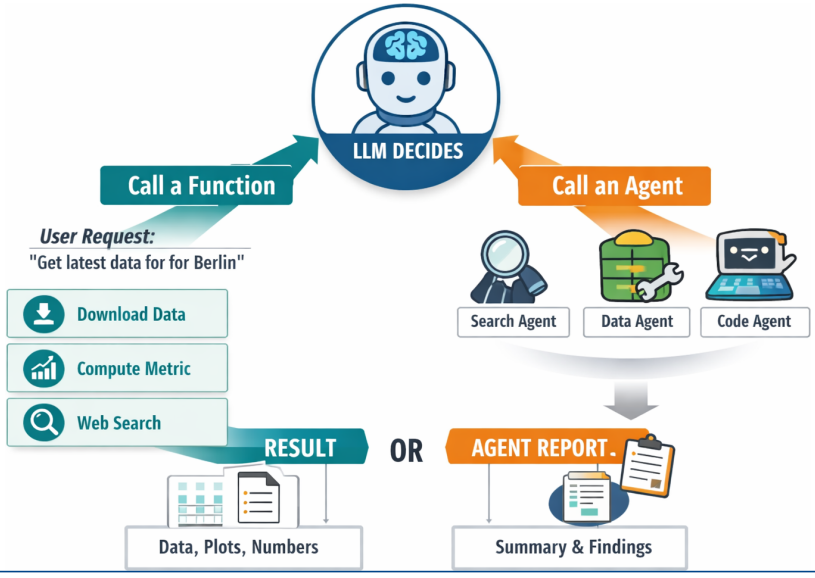
- clear capabilities: what is possible
- safety boundary: what is not allowed
- reproducible results: same tool ⇒ same behavior

Think of it as a menu: the model chooses an item, the system cooks it.

The tool list becomes the model's action space.

Tool registry (provided to the model)

```
TOOLS = [
  {
    "name": "get_weather_forecast",
    "description": "Retrieve ICON-EU forecast valu
    "inputs": {
      "location": "string",
      "variable": "t2m|wind|precip",
      "lead_time_h": "integer"
    }
  },
  {
    "name": "google_search",
    "description": "Search the web for up-to-date
    "inputs": { "query": "string" }
  }
]
```


LLM DECIDES
Call a Function
Call an Agent
User Request:
"Get latest data for for Berlin"
Download Data
Compute Metric
Web Search
Search Agent
Data Agent
Code Agent
RESULT
OR
AGENT REPORT.
Data, Plots, Numbers
Summary & Findings

Tools can be Functions *or* Agents

Next step beyond tool calling

With function calling, a "tool" does not need to be a single function. It can also be an agent — a specialist that performs multi-step actions.

Example specialist agents

- **Search agent:** finds reliable sources
- **Data agent:** downloads + preprocesses datasets
- **Code agent:** runs scripts and produces figures
- **Report agent:** writes a structured summary

Key idea: the LLM becomes a coordinator that delegates tasks and combines results into one coherent answer.

Example: delegate to a specialist agent

```
{
  "tool_name": "search_agent",
  "arguments": {
    "task": "Find the latest DWD warnings for Be
    "sources": "official",
    "return_format": "short_summary + links"
  }
}
```

Delivers as return:

```
{
  "agent_result": {
    "summary": "...",
    "sources": ["...","..."]
  }
}
```

LLM Orchestration: Decisions & Guardrails

LLM = Decision Engine

For each user request, the model decides what happens next:

- answer directly (when trivial)
- call a function tool (fast action)
- delegate to an agent (multi-step work)
- ask a clarifying question (missing info)

Key idea: The model routes the task to the best capability.

(Like a dispatcher: choose the right action at the right time.)

Guardrails = System Control

The system stays in charge of execution:

- only approved tools exist (tool registry)
- arguments are validated (schemas)
- execution runs outside the LLM
- logging & reproducibility by design

Result: Reliable actions, not hallucinated actions.

The LLM proposes — the backend disposes.

The Race for Function & Agent Standards

OpenAI — Tool Calling + JSON Schema

- tools defined via JSON Schema
- Structured Outputs / strict schema for reliable calls

Anthropic — MCP (Model Context Protocol)

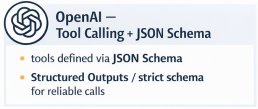
- open connector: model ↔ MCP server ↔ tools/data
- goal: universal plug-in interface for enterprise tools

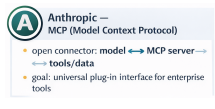
Google — Gemini Function Calling

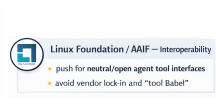
- built-in function calling in the Gemini API ecosystem
- tight integration with the Google stack

Linux Foundation / AAIF — Interoperability

- push for neutral/open agent tool interfaces
- avoid vendor lock-in and "tool Babel"







Our Opportunity: DAWID Puts Trusted Functions at Our Fingertips

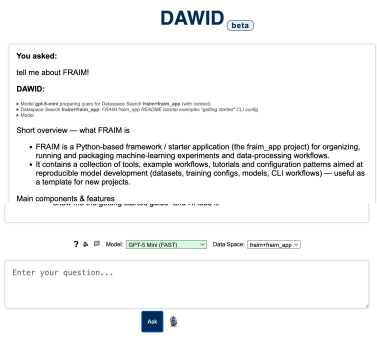
What we can do immediately

DAWID gives us a practical advantage: we can expose our own operational capabilities as tools.

- tools = trusted ECMWF / DWD / NHMS functions (not generic chat)
- each tool encodes a workflow step we already know
- the LLM becomes the orchestrator and UI

Result:

- faster exploration and analysis
- reproducible results (same tools, same outputs)
- knowledge transfer: tools carry expertise



DAWID: User Interface

— OUR functions!!

AI Forecasting: Learning to Predict the Next Weather State

Key idea

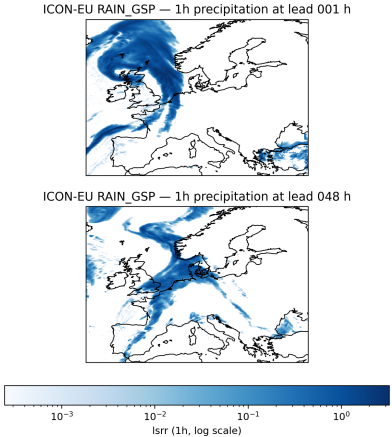
- The atmosphere has a **state** (a snapshot)
- Forecasting means predicting a **future state**
- A neural network learns a mapping:

$$x(t) \quad \Rightarrow \quad x(t + 48h)$$

where x contains fields like temperature, wind, pressure, humidity.

Training: learn from millions of examples (reanalysis + observations).

After training, the model can produce forecasts extremely fast.



Neural Network Forecasting: One Step → Many Steps

How an AI forecast is produced

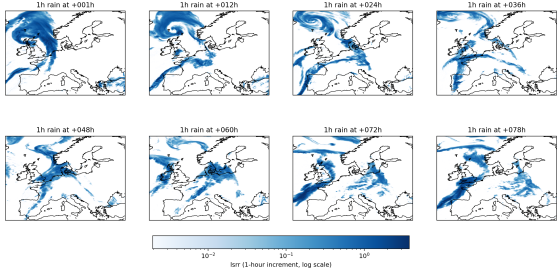
- The model learns: state now → state later
- Longer lead times come from **repeating the prediction step**

$$x(t) \to x(t + \Delta t) \to x(t + 2\Delta t) \to \dots$$

Physics detail: accumulated variables

- precipitation is often stored as an **accumulated sum**
- to get **1-hour rain** at lead L:

$$RAIN(L) - RAIN(L-1)$$



Example: ICON-EU 1-hour precipitation for multiple lead times.

Here: Nowcasting or NWP - depending on input, time scales and variables!

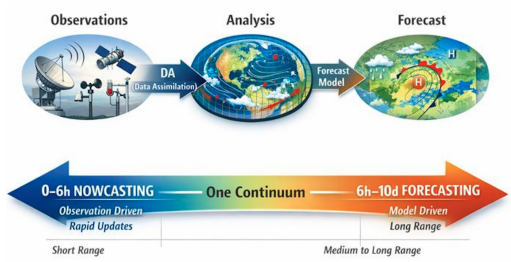
Nowcasting and NWP: One Continuum (Observations → DA → Forecast)

NWP with Data Assimilation (DA) is the full framework

- **observations** enter continuously:
  radar, satellite, aircraft, surface stations, etc.
- **DA** combines obs + model:
  best estimate of the 3D atmosphere at "now"
- then **forecast propagation** produces future weather

Nowcasting is the short-range regime

- lead times: **minutes to a few hours**
- very high weight of recent observations
- focus on rapidly evolving phenomena:
  convective storms, precipitation cells



AI can support different parts: observations, DA, model emulation, and products.

There is no strict boundary: it is one continuum.

Forecasting in One Sentence

Weather forecasting = state estimation + prediction

- We estimate the **current atmospheric state**
- Then we predict how it changes in time

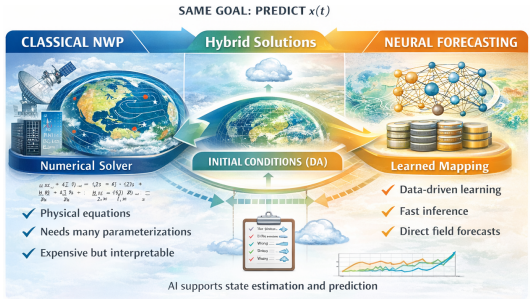
$$\underbrace{x(t_0)}_{\text{best estimate now}} \quad \rightarrow \quad \underbrace{x(t_0 + \Delta t)}_{\text{future state}}$$

Why it is hard

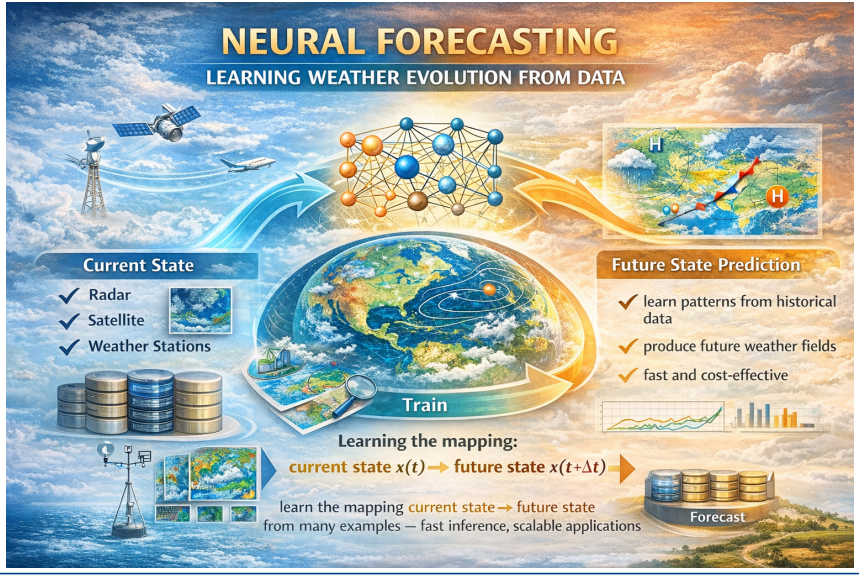
- the atmosphere is **chaotic**
- small errors grow with time
- observations are incomplete and noisy

So we quantify uncertainty

- ensembles: many forecasts instead of one

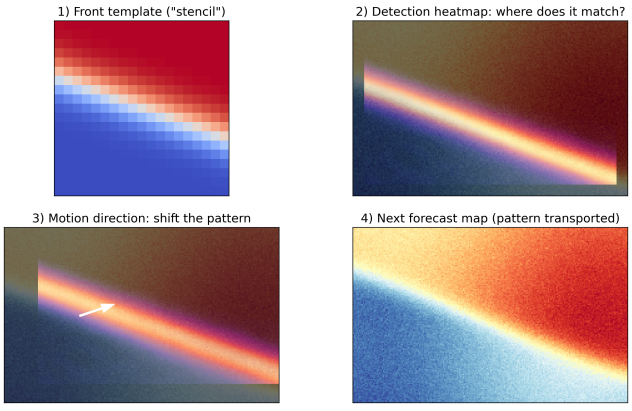


Forecasting: observation → analysis → prediction


NEURAL FORECASTING
LEARNING WEATHER EVOLUTION FROM DATA
H
H
Current State
Radar
Satellite
Weather Stations
Train
Future State Prediction
learn patterns from historical data
produce future weather fields
fast and cost-effective
Learning the mapping:
current state x(t) → future state x(t+Δt)
learn the mapping current state → future state
from many examples — fast inference, scalable applications
Forecast

How a Neural Network Learns Forecasting: Front Template + Motion

Neural forecasting intuition: template → heatmap → shift → next field



Visual argument: the NN learns **templates** (front patterns) and how they **move** / **evolve** over time.

One Step, Many Steps: Templates + Transport + Accumulations

Neural forecasting learns reusable building blocks

- learn **templates** for structures:
  fronts, rain bands, vortices, jets
- learn **how they move and change** from context
  (e.g. wind, humidity, stability)

Then: one step → many steps

$$x(t) \to x(t + \Delta t) \to x(t + 2\Delta t) \to \dots$$

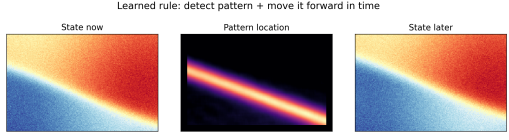
Important for users: accumulated products

- many outputs are stored as **accumulations**
- 1-hour precipitation at lead L is:

$$RAIN(L) - RAIN(L-1)$$

Take-away:

Forecasting = detect patterns + transport them forward in time.



Templates → heatmaps → shifted patterns → next map (repeated for longer lead times).

A Minimal Forecasting World: A Blob Moving on a Circle

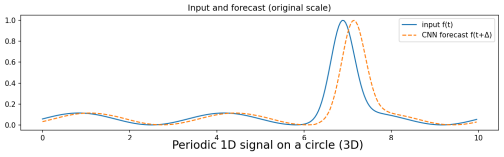
We create a simple 1D world on $[0, 10]$:

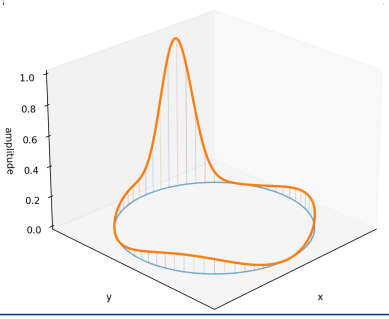
- a function $f(t, x)$ (a "blob")
- periodic domain: $x = 0$ connects to $x = 10$
- dynamics: pure translation

$$f(t + \Delta, x) = f(t, x - \delta)$$

Forecasting task: given $f(t, \cdot)$, predict $f(t + \Delta, \cdot)$.

This captures the essence of advection: structures move.





What the Neural Network Sees

Important: the input is the full signal, not single points.

The network gets all grid values of $f(t, x)$ at once:

$$f(t, \cdot) \in \mathbb{R}^N \qquad (N = 256)$$

But the computation is local: each output point is computed from a neighborhood.

Kernel size k = width of the filter

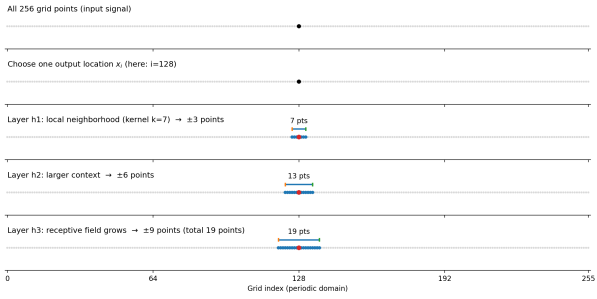
With $k = 7$, the network looks at:

$$[x_{i-3}, \ldots, x_i, \ldots, x_{i+3}]$$

Stacking layers increases context

After 3 conv layers (all $k = 7$), each output point depends on roughly:

$$1 + 3 \cdot (k - 1) = 19 \text{ grid points}$$

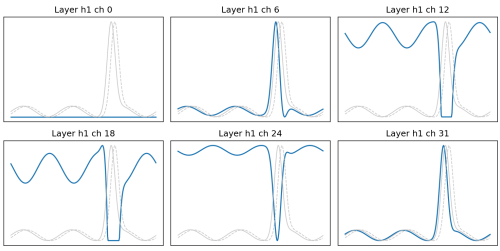


Inside the CNN: Many Feature Variables

The CNN transforms the input into many internal variables:

$$1 \to 32 \to 32 \to 32 \to 1$$

- input: one function $f(t, x)$
- hidden layers: 32 feature channels each
- output: predicted function $\hat{f}(t + \Delta, x)$



Example: 6 of 32 feature channels in layer h1.

Interpretation: each channel is a learned detector for local shapes (edges, slopes, curvature, . . .).

The variables h_1, h_2, h_3 are hidden feature maps inside the CNN. They form a latent feature space: not observed, not physical, but learned. Each layer contains many channels (e.g. 32), extracting different local aspects of the signal. The forecast emerges by transforming $f(t, \cdot) \to h_1 \to h_2 \to h_3 \to \hat{f}(t + \Delta, \cdot)$.

Why This Is *Not* Explicit Pattern Matching

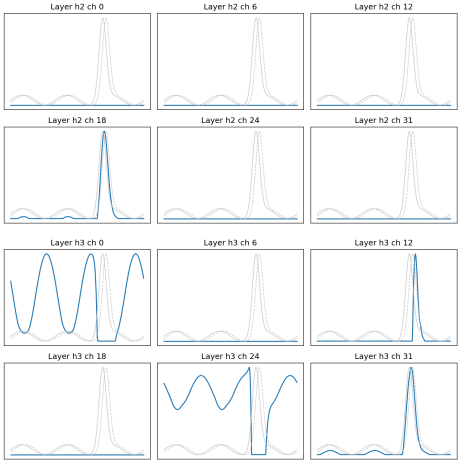
A common misconception is:

> *"The network finds the blob pattern and just copies it forward."*

What actually happens:

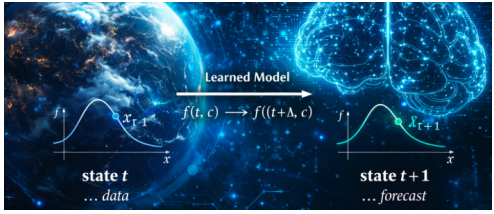
- the CNN computes many internal feature variables (32 channels)
- each channel responds to different local aspects
- the forecast is produced by recombining these features

Key point: the NN does not store templates — it learns a continuous mapping from input to forecast.



Example: 6 of 32 feature channels in layer h3.

What We Can Do Already



We already have strong building blocks in place:

- We have created a framework for AI-based weather forecasting with Anemoi
- We have developed libraries (mfai, FRAIM) for:
  - downscaling and high-resolution products
  - road weather services
  - high-impact weather feature extraction
  - weather interpretation and explainability
  - nowcasting of observation fields (radiation, precipitation, . . .)
- We have gained expertise across the full AI/ML value chain in weather services and international organizations.
- We have created networking and structures for joint development and sustainable research — connected to leading global AI/ML developments.

What We Can Do Already (Operational AI Forecasting)

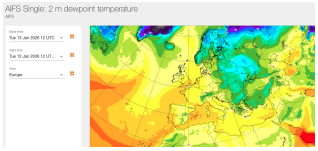
AI forecasting is already operational.
In Europe, several AI-based forecasting systems are now in production use (e.g. AIFS, AICON, BRIS). They complement classical NWP by providing:

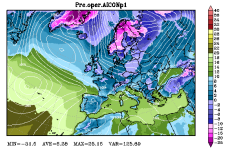
- fast state-to-state forecasts (minutes instead of hours)
- competitive large-scale skill for key variables
- robust baselines and rapid experimentation

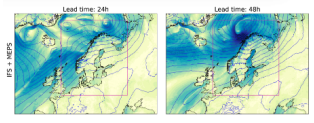
Key message: AI is no longer "research only" — it is part of the operational toolbox.

This opens a clear opportunity:

- combine NWP + AI hybrids
- integrate new observation-driven products
- accelerate development cycles via ML pipelines







Library Ecosystem: Complementary Capabilities

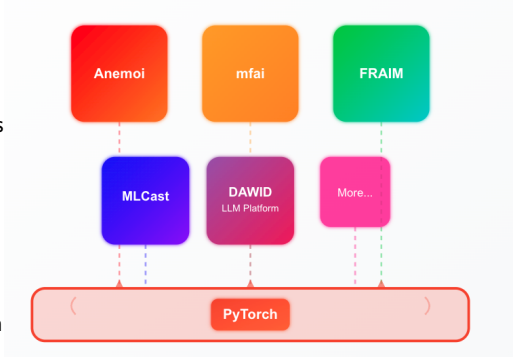
We are building a strong ecosystem of AI libraries
Different libraries contribute complementary capabilities:

- **Anemoi:** end-to-end AI weather models
- **mfai:** vision transformers and many applications
- **MLCast:** nowcasting library, observation based
- **FRAIM:** products and services, full value chain

Positive outlook

- no single library needs to do everything
- modular building blocks enable rapid innovation
- shared standards ⇒ interoperability and reuse

Key message: Together, these components form a platform for scalable AI in weather services.



Ecosystem view: different strengths, one common mission.

Why Physics Matters: Future AI Forecast Systems Will Be Physics + AI

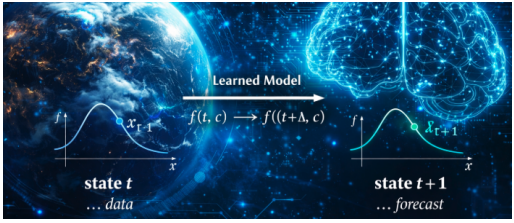
Weather is not just a pattern problem

Forecasting is about consistent evolution of a physical state:

- conservation laws (mass, energy, water)
- balances and constraints (geostrophy, stability)
- multiscale interactions (local storms ↔ global flow)

Why "pure AI" can fail

- excellent short-term skill can still drift long-term
- small violations accumulate (mass / moisture / energy)
- rare extremes need physics consistency, not just averages



AI gives speed and learning.

Physics gives truth and trust.

- AI learns corrections / closures
- constraints and invariants built into the model
- physics-based evaluation & reliability

What You Can Do (A): Use LLMs as a Tool — You Stay the Master

Use AI for: understanding, exploring, developing

Think of LLMs as a power tool for knowledge work:

- explain unfamiliar concepts in your context
- explore alternatives and trade-offs quickly
- generate drafts, code sketches, slide structure
- summarize documents, meetings, research threads

The right mindset

- you are the pilot, the AI is the assistant
- ask for options, then decide yourself
- validate important facts (sources, experiments)

Key message: AI needs guidance — quality comes from your questions and your checks.

Human judgment stays in control.

What LLMs Do Today — and What "World Models" Add Next

LLMs today (what they actually do)

- generate text/code by next-token prediction
- strong at language tasks: explain, summarize, draft, transform
- can appear to reason, but truth not guaranteed
- without grounding: can be confidently wrong

World-model approaches (the next step)

- learn explicit state representations of a system (world state)
- learn dynamics: state(t) + action $\rightarrow$ state(t+1)
- enable planning, long-horizon consistency, and controllable actions
- usually needs grounding (data, sensors, simulators, tools)

Message: LLMs are powerful interfaces — world models aim at reliable dynamics.

LLMs for knowledge work

Be careful:

- verify facts, numbers, and operational details
- demand sources or reproduce with scripts
- watch for missing assumptions and edge cases

Take full advantage:

- accelerate understanding and documentation
- structure tasks into steps, checklists, and options
- generate first drafts and improve clarity
- connect to tools/agents for grounded actions

Fast thinking + human validation = real productivity.

What You Can Do (B): Rebuild Services with Tools, Agents, and Clear APIs

Use the AI revolution to modernize services

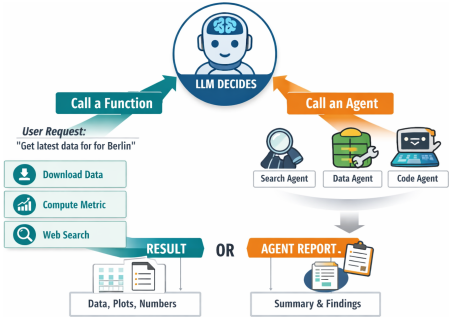
LLMs work best when they can call clear interfaces. This is an opportunity to rebuild our services in a clean way:

- define small, robust functions (tools)
- combine them into agent workflows
- expose everything as API-based services

Why this is powerful

- clarity: inputs/outputs become explicit
- reuse: one tool serves many applications
- automation: agents connect tools: end-to-end
- acceleration: faster prototyping and delivery

Key message: LLMs need good tool definitions — we can use this to create better services for ourselves.

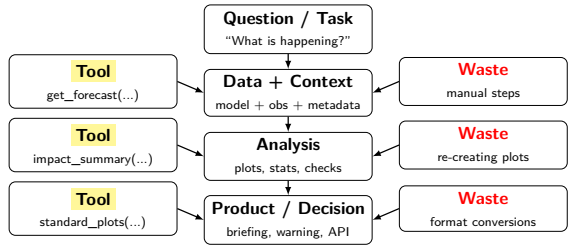


Tool calling turns services into reusable building blocks.

Slide 39: Rebuild Services — Analyze Work, Find Waste, Build Tools

Step 1: Look at our daily workflow

- Where do we repeat the same steps?
- Where do we copy/paste between systems?
- Where do we wait for data, plots, approvals?
- Where do we lose context (emails, chats, files)?



Replace repeated manual work by reusable tools with clear interfaces.

Step 2: Turn waste into tools

- define a clear input/output interface
- make it callable (API / function / agent tool)
- reuse it everywhere: DAWID, scripts, services, pipelines

Key message: If a task repeats, it should become a tool.

What You Can Do (C): Bring Your Domain Expertise — Make AI Your Own

AI becomes powerful when it is grounded in meteorology

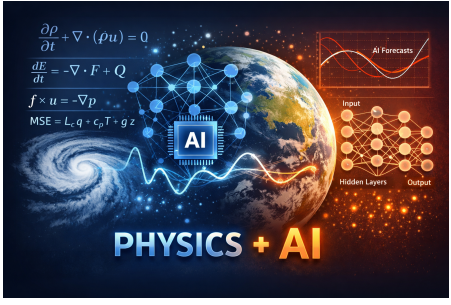
LLMs and neural networks do not automatically understand:

- processes, states, and transitions
- limits and plausibility
- physical constraints and balances
- weather regimes and rare extremes

Your expertise is the missing ingredient

- define what matters: variables, events, diagnostics
- define what is allowed: limits, physics, consistency
- define what is useful: products, warnings, explanations

Key message: Do not outsource AI — embed your knowledge into tools, workflows, and constraints.



Domain expertise turns AI into trust.