

— IE437 · DATA-DRIVEN DECISION MAKING AND CONTROL — LECTURE 0

Data-Driven Decision Making and Control

Deciding well under uncertainty — three criteria, four categories, four real systems

Jinkyoo Park · KAIST — Industrial & Systems Engineering

The map — how every chapter fits one story, and what each one decided in the field

— 01

The one question

Every lecture this term is the same question, asked again with one more thing made harder.

Learning route — identify the decision before naming the method

Bring: No algorithm knowledge is assumed here.

FIRST PASS	WHAT TO DO
Follow the idea	Decision → information → one choice or a sequence → evidence
Work without the solution	Identify the action, objective and available feedback in the temperature example.
Return later	The detailed industry architectures and research outlook are reference cases.

For the temperature thread: **predict** → **calculate** → **reveal and check** → **change one condition**. Complete the core calculation before reading the research extensions.

What this course is about — in one sentence

How do we make **good decisions under uncertainty?**

That is the entire course. We ask it under changing assumptions: a known objective, an uncertain model, decisions through time, and eventually a fixed dataset. Known-model chapters establish tools before we learn from experience. Multi-agent games are a preview of IE579.

Two disciplines answer it differently, and this course lives where they meet — **decision-centric AI**: use data and learning to make high-dimensional decisions that classical optimisation alone could not reach.

Temperature thread — name the decision before choosing a method

shared teaching example · predict before revealing the calculation

A room is at 20°C and the target is 22°C. Let $x = T - 22 = -2$. A normalized heater command $u \in [0, 2]$ gives $x' = x + u$; the one-step cost is $c = (x')^2 + u^2$. This is a small teaching model, not calibrated building physics.

Predict: Does the largest heating command necessarily give the best decision?

Calculate and check.

COMMAND	NEXT TEMPERATURE	COMFORT ERROR + EFFORT
$u = 0$	20°C	$4 + 0 = 4$
$u = 1$	21°C	$1 + 1 = 2$
$u = 2$	22°C	$0 + 4 = 4$

The decision is the heater command; the measured temperature is information. A target temperature alone does not specify how to trade comfort against effort.

Try it — the temperature response is unknown

work independently · reveal only after writing an answer

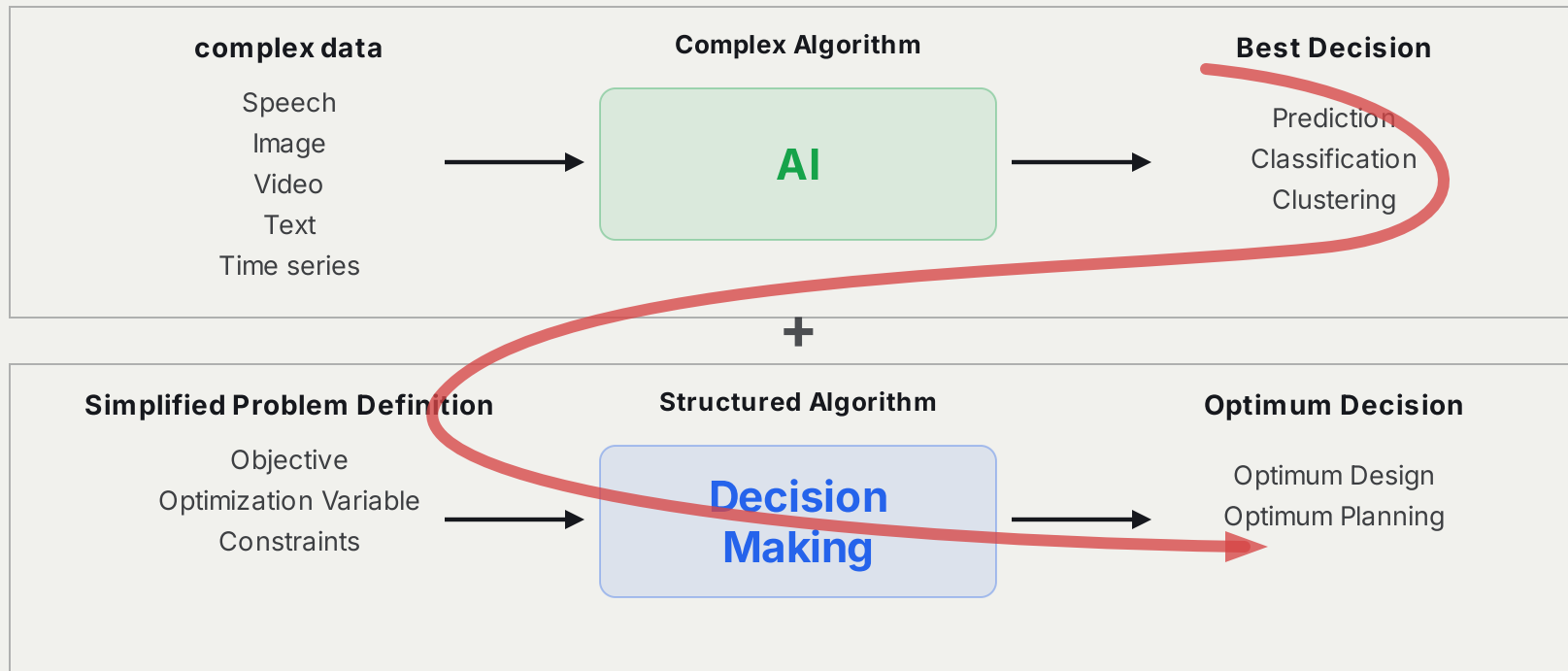
Keep the same goal and heater, but suppose the relation between command and next temperature is unknown. Name what you would measure, what you would learn, and how you would evaluate a decision.

Check your answer. Measure command–temperature transitions and their cost. Learn a response model or learn a value/policy from experience. Evaluate the resulting comfort–effort cost on new allowed trials. If trials are unavailable, evaluation must use the existing log and explicit assumptions.

Across the course, keep asking what is known, what is learned and whether another experiment is allowed.

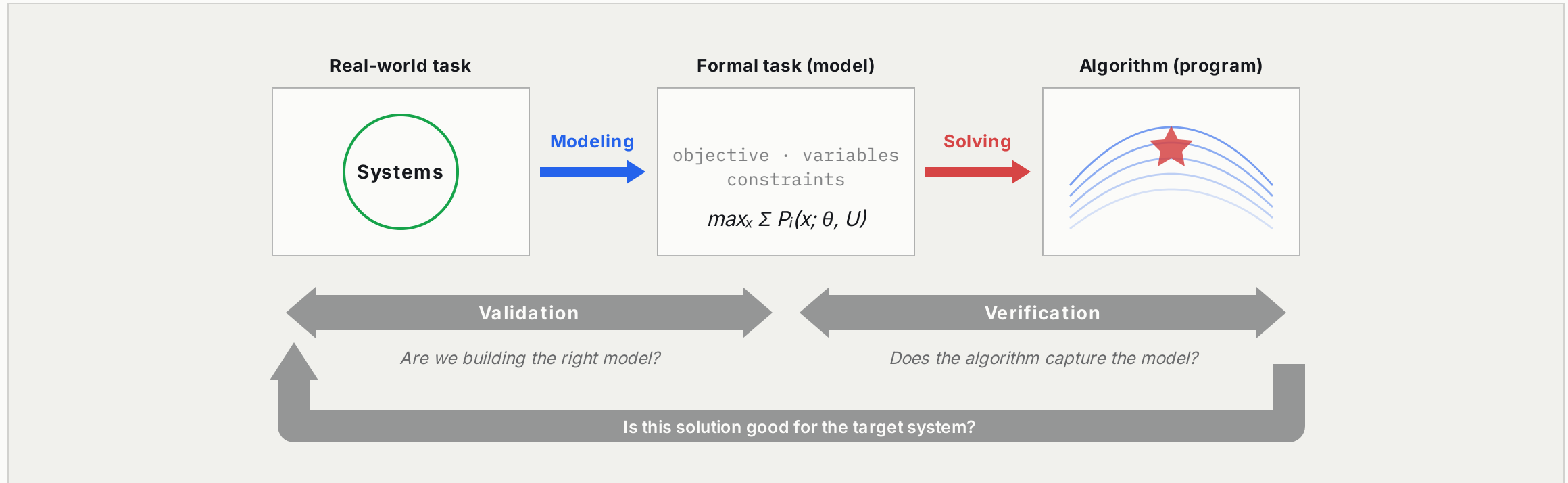
AI, decision making, and what joins them

TWO PIPELINES, AND WHAT JOINS THEM



Decision-centric AI. Feed what AI learns from data into the decision problem, and high-dimensional decisions become reachable that neither could reach alone.

From a real system to a solved problem



Formulation and solution are two different arrows, policed by two different questions. **Data helps on the left arrow too** — a model fitted to data represents the world more faithfully, not merely faster.

Two kinds of uncertainty — name them once

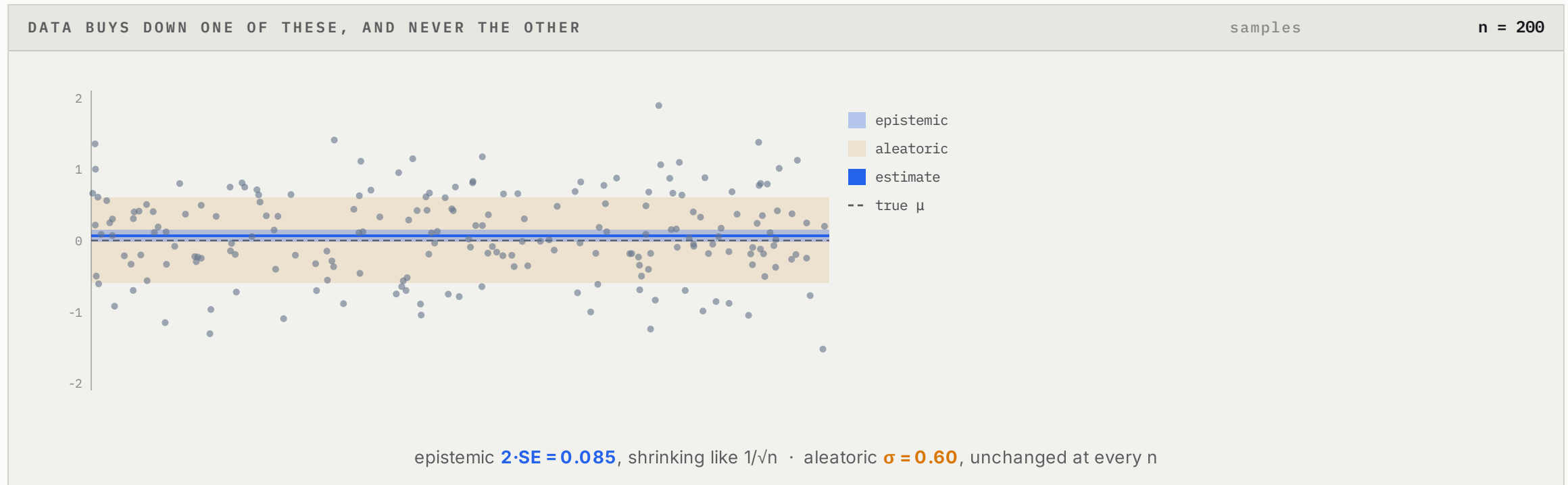
Everything we model is uncertain in one of two ways. The distinction recurs all term.

-
- **Aleatoric** (statistical) uncertainty — the world is genuinely stochastic. Repeating the experiment gives different outcomes. More data alone does not remove this variability under the chosen observation model; improved sensing or a changed process can alter it.
-
- **Epistemic** (model) uncertainty — we simply *don't know* the model: its parameters, its state, its dynamics. Relevant data can reduce it; misspecified models may remain confidently wrong.
-

The whole course is a campaign against **epistemic** uncertainty.

Learning the parameter (Ch 2), the structure (Ch 3), the function (Ch 4), the design map (Ch 5–6), the dynamics (Ch 8–11) — while *respecting* the aleatoric uncertainty that no data can erase.

The two, drawn — only one of them answers to data



Drag the sample count. The **blue band collapses** like $1/\sqrt{n}$ — that is what data buys. The amber band is the world's own spread, and it does not move at all however many samples you take.

— 02

Three criteria, four categories

Decision making under uncertainty, split by three questions — and the four cells this course actually visits.

The three criteria

p. 8 of the source deck — the sentence the whole taxonomy hangs on

"Decision making under uncertainties." Three questions divide every method in the zoo, and the answers are independent of one another.

#	THE CRITERION	ONE END	THE OTHER END
1	Is the model known?	model-based — handed to us	data-driven — learned from data
2	How many stages ?	static — one decision	dynamic — a sequence
3	How many decision makers ?	single agent — an optimum	many agents — an equilibrium

Criterion 1 is where the two uncertainties live: epistemic uncertainty is exactly *not knowing the model*, and it is the axis this course is named after. Criteria 2 and 3 say what shape the problem has once you do know it.

One familiar decision, three versions — stocking a shop

Suppose selling one item earns 5, ordering it costs 2, and unsold items have no value. Profit is $5 \min(q, D) - 2q$.

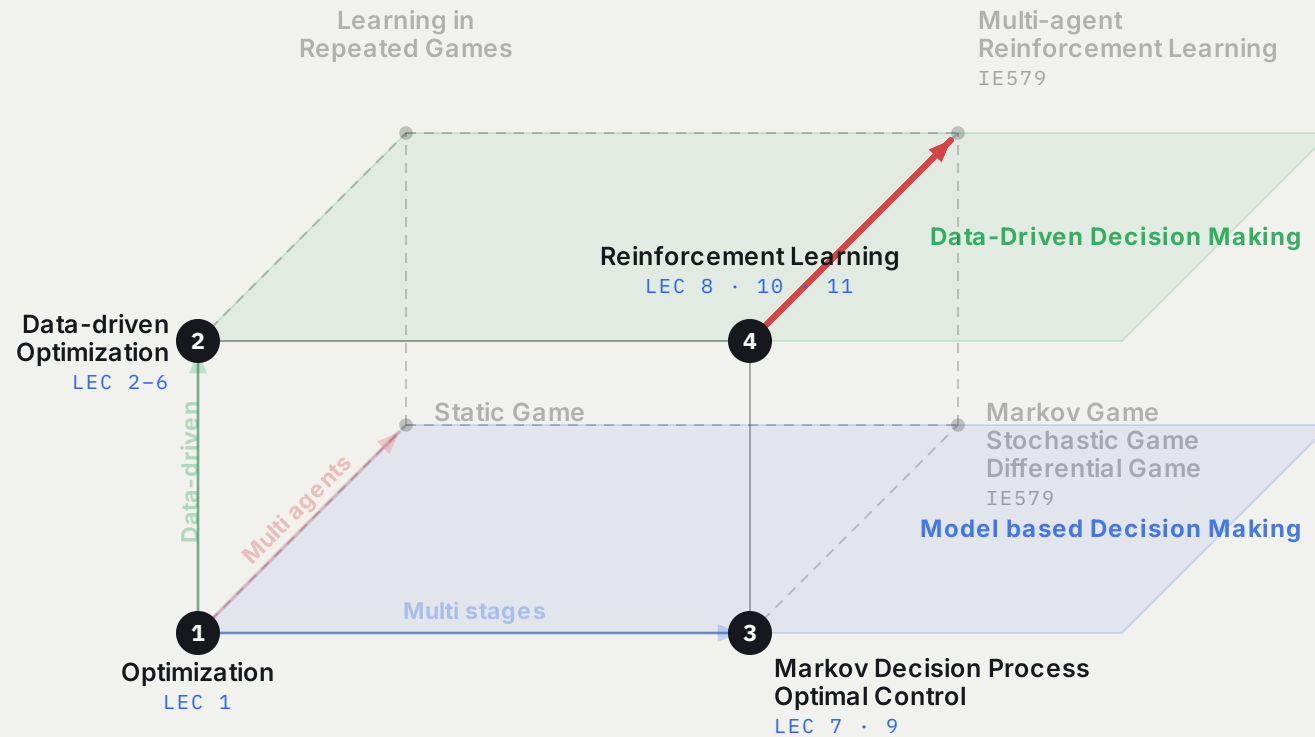
SETTING	WHAT MUST BE DECIDED	WHAT IS NEEDED
demand is known to be 10	today's order q	compare profits: $q = 8, 10, 12$ gives 24, 30, 26
demand is uncertain	today's order before seeing demand	a demand distribution; compare expected profit
orders repeat daily	order based on stock and new information	a policy; today's stock affects tomorrow

The product is the same. Changing **information** and **time** changes the decision problem. Start with this distinction before learning the names of algorithms.

Three criteria make a cube

THE CUBE — EVERY LECTURE, ONE MAP

STEP 6 / 6



The far face is not ours. Multiple decision makers introduce **strategic interaction** and solution concepts such as **equilibria** — the subject of IE579, Game Theory and Multi-Agent RL. This course stops at the near face.

The origin is plain optimisation. Step through the numbered route — **step ④ is the one to watch**, where the data-driven axis is crossed a second time and the count of unknowns doubles.

Criterion 1— model-based becomes data-driven

the deepest axis, and the course's namesake

Two ways to know the world you decide in:

- **Model-based** — the objective, constraints, transition and dynamics are *given*. You compute the optimum. (*Lectures 1, 7, 9*)
- **Data-driven** — you have only *data*. You must learn enough of the world to act: a belief (Ch 2–4), a surrogate (Ch 5–6), a value or a policy (Ch 8, 10, 11).

THE MOVE THAT DEFINES THE COURSE

Take a model-based method, **delete the model**, and replace it with data.

optimisation → Bayesian optimisation DP → value-based RL optimal control → policy-based RL

This is a teaching map. In standard RL terminology, **model-based RL can itself be data-driven** when it learns a model, as in Lecture 11.

Criterion 2 — static becomes dynamic

	SINGLE AGENT	MULTI AGENT
Static	optimisation	static game
Dynamic	dynamic optimisation / control	dynamic game

Static — one decision. You choose x once; the modeled decision is chosen once. BO may query repeatedly to find that one design. This is classical optimisation (Lecture 1) and its uncertain, data-driven cousins (Lectures 2–6).

Dynamic — a sequence. Each decision reshapes the state the next one faces. This is the realm of MDPs, optimal control and reinforcement learning (Lectures 7–11).

The single optimisation splits into time — and **the value function is born**.

Criterion 3 — one decision maker becomes many

	OPTIMISATION	GAME
single objective	one optimum x^*	—
coupled objectives	—	equilibrium (Nash, ...)

When other agents — also optimising, also learning — share the world, the very notion of "optimal" changes. There is no single best action independent of what the others do. The solution concept shifts from an **optimum** to an **equilibrium**.

This course stops before that axis. Crossing it is the subject of the follow-on course, [IE579 Game Theory and Multi-Agent Reinforcement Learning](#) — and the cube above shows exactly which face it takes over. Naming the axis here is what makes the boundary of this course legible.

The four categories, and the four systems that prove them

pp. 13–16, 19, 57–58 of the source — the badges ①②③④ you will see all term

	CATEGORY	WHAT IS GIVEN	LECTURES	THE SYSTEM IT DECIDED
①	Fundamentals — static, model-based	the model	1–3	<i>the template every other cell inherits</i>
②	Data-driven static decision making	data only	4–6	wind farm layout · traffic signal split
③	Fundamentals — dynamic, model-based	the model, over time	7, 9	<i>the template Part IV deletes</i>
④	Data-driven dynamic decision making	data only, over time	8, 10–12	semiconductor furnace · offline meta control

The rest of this lecture walks the four categories — and shows, for each, [a system where the decision was actually made](#).

Check — reading the cube

The course maps every decision problem onto three axes. Which three?

- ☒ A One decision or a sequence · the world given as a model or learned from data · deciding alone or against others
- ☐ B Discrete or continuous · linear or non-linear · convex or non-convex
- ☐ C Deterministic or stochastic · finite or infinite horizon · one agent or many
- ☐ D Supervised, unsupervised, or reinforcement · online or offline · tabular or approximate

Stages, model, and agents. Every lecture in this course is one position on that cube, and the course as a whole is a **route** through it: the chapters compare different assumptions. The location helps identify candidate methods; the details of the task determine which are appropriate.

— 03

Categories ① and ② — the static half

One decision, made once. First with the model in hand, then with nothing but data — and two systems that were decided this way.

① Fundamentals — optimisation, and belief

Lectures 1–3 · static, model-based

Given a model, a decision is a program:

$$\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0, \quad h(x) = 0$$

Make the model *uncertain* rather than absent and the same program acquires a belief. Bayes' rule is how the belief moves:

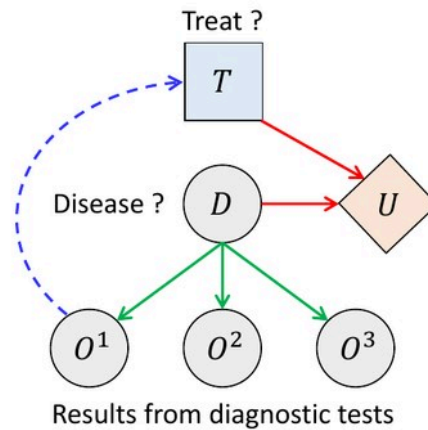
$$p(\theta \mid \text{data}) = \frac{p(\text{data} \mid \theta) p(\theta)}{p(\text{data})}$$

Lecture 2 carries the belief over one parameter; Lecture 3 gives it structure, and then adds decision and utility nodes — at which point the graph *is* a one-stage decision problem.

From a Bayesian network to a decision

p. 18 of the source — a chance node, a decision node, a utility node

Bayesian Network + **Decision node** + **Utility node** = **Decision network (Influential Diagram)**



T	D	$U(T, D)$
0	0	0
0	1	-10
1	0	-1
1	1	-1

→ Conditional edge
 → Functional edge
 ----> Information edge
 (often omitted)

○ A **chance node** corresponds to a random variable

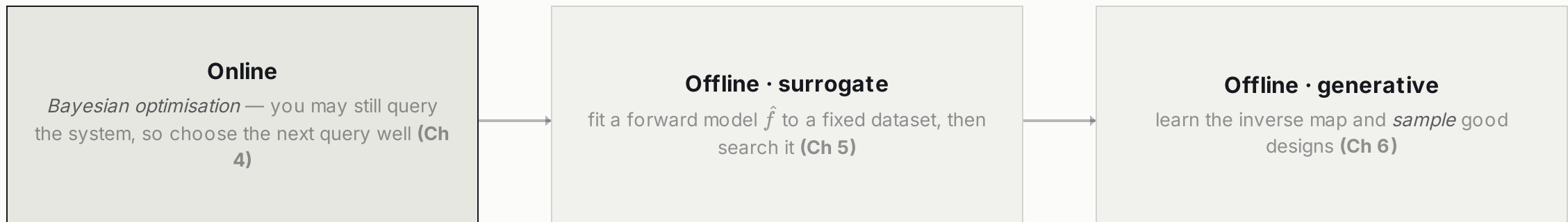
□ A **decision node** corresponds to each decision to be made

◇ A **utility node** corresponds to an additive utility component

Add a **decision** node and a **utility** node to a Bayesian network and you have an **influence diagram**. Lecture 7 adds repeated decisions, a Markov state and an explicit transition model to build an MDP.

② Data-driven static — three routes when the model is gone

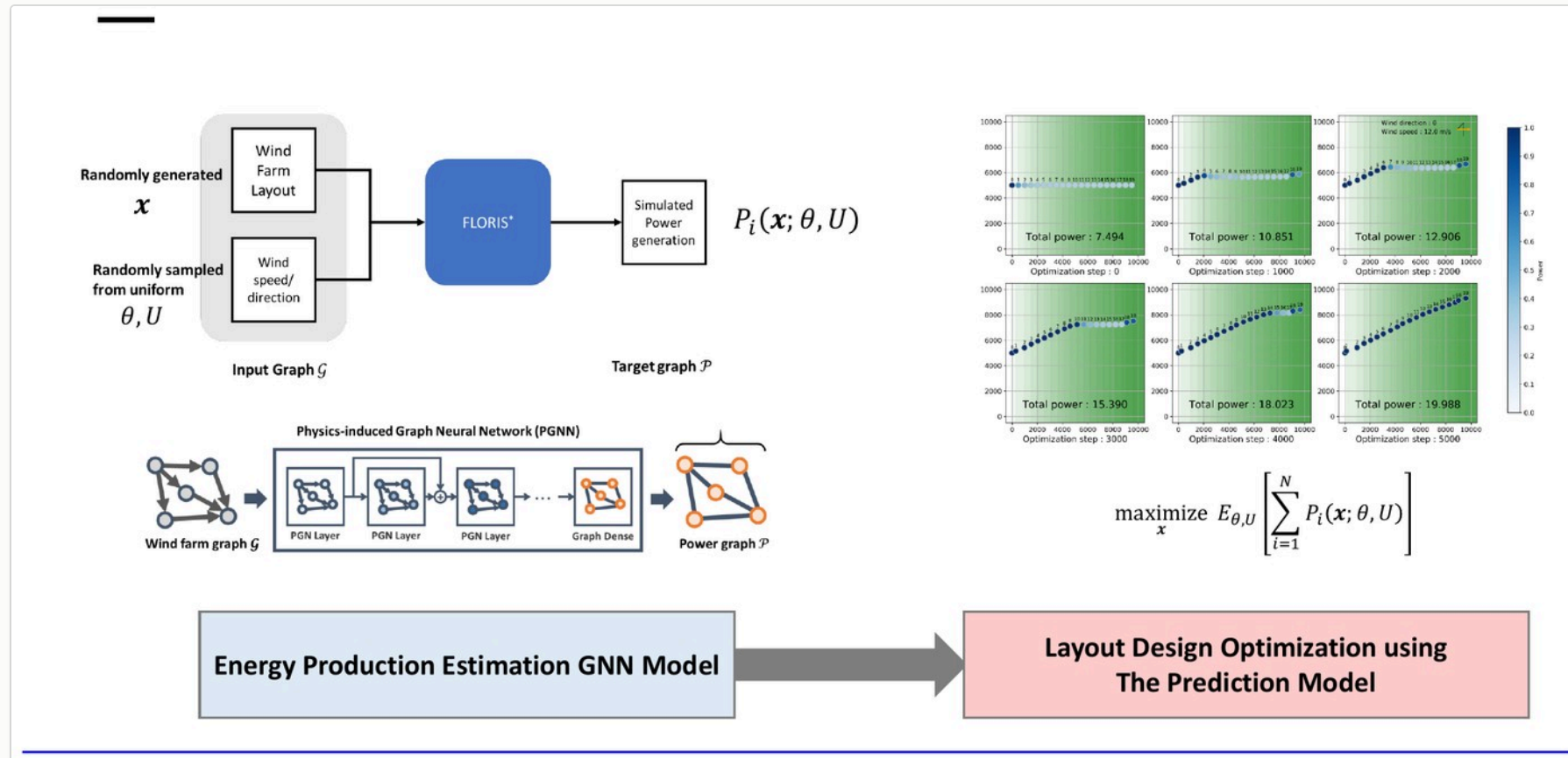
pp. 19 and 32 of the source · static, data-driven



A research extension is **offline meta optimisation with online adaptation** — learn across many past tasks, then adapt to today's in a handful of trials. The traffic-signal case below is exactly this.

Case A · Wind farm layout

Application Example 1 — where do N turbines go, when each steals wind from the next?



The decision is $\mathbf{x}$, the turbine positions. The obstacle is that power $P_i(\mathbf{x}; \theta, U)$ has no closed form: turbines sit in each other's **wake**, so the objective couples every pair.

Case B · Traffic signal split

Application Example 2 — offline meta data-driven optimisation



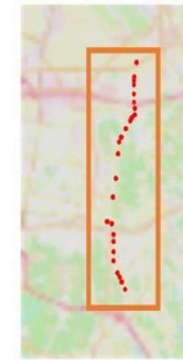
(a) Hangzhou, China



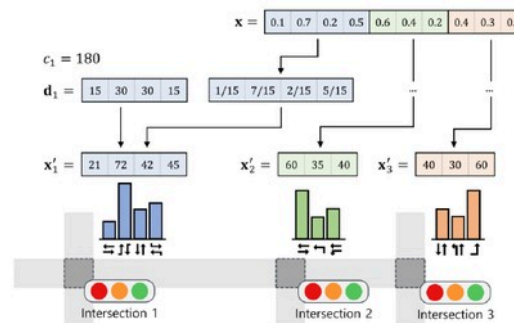
(b) Manhattan, USA



(c) District A



(d) District B



The same decision — the green split at every intersection — in Hangzhou, Manhattan and two synthetic districts. **Every intersection is a new task**, and you cannot run a thousand trials on a live junction.

Check — what makes a decision problem hard

What extra question does a decision problem ask beyond predicting an outcome?

- ☐ A How can we guarantee that more data always improves every decision?
- ☒ B Which allowed action best serves the objective, given its consequences and the information available?
- ☐ C Why must every decision problem be nonconvex?
- ☐ D Which neural network is required for every decision?

A demand prediction is an input to an inventory decision. Choosing inventory also requires costs, constraints and a model or evidence about consequences. In logged interaction data, only the chosen action's outcome is observed; a simulator or known model may provide additional information about alternatives.

— 04

Categories ③ and ④ — the dynamic half

Now the decision is a sequence, and each one reshapes the state the next one faces.

③ Fundamentals — two scientific parents

Lectures 7 and 9 · dynamic, model-based

	LINEAGE A · OR / DYNAMIC PROGRAMMING	LINEAGE B · CONTROL THEORY
MODEL-BASED (ORIGIN)	MDP & Dynamic Programming LECTURE 7	Optimal Control / Planning LECTURE 9
DATA-DRIVEN (EXTENSION)	Value-Based RL LECTURE 8	Policy-Based RL LECTURE 10

We teach two useful routes into RL: **value-based** and **policy-based** methods. They overlap in actor–critic algorithms.

- **Value-based RL** is *dynamic programming with the model deleted* — discrete, value-centric, Bellman's operations research.
- **Policy-based RL** is *optimal control with the dynamics deleted* — continuous, control-law-centric, Pontryagin and Kalman's control theory.

The same claim, in two grids

pp. 57–58 of the source deck — where Lectures 7 to 10 come from

Zoom into the dynamic, single-agent cell and split it twice: by **action space** and by **time**. Handed a model, the four boxes are the classical theories.

MODEL-BASED	FINITE ACTION SPACE	INFINITE ACTION SPACE
discrete time	discrete-time MDP · $P(s_{t+1} \mid s_t, a_t)$	discrete-time dynamic system · $x_{t+1} = f(x_t, u_t)$
continuous time	continuous-time MDP · $P(s_{t+h} \mid s_t, a_t)$	continuous-time dynamic system · $\dot{x}_t = f(x_t, u_t)$

For this course’s main examples, deleting the given model leads to the methods below. These are not exclusive categories: policies can be discrete and value functions continuous.

MODEL-FREE	FINITE ACTION SPACE	INFINITE ACTION SPACE
discrete time	value-based reinforcement learning	policy-based reinforcement learning

④ The doubling — why the second crossing is not the first again

FIRST CROSSING • LECTURE 1 → 2

The static world. In our first examples, the objective is the main unknown:

$$\min_x f(x) \quad \text{s.t.} \quad g(x) \leq 0$$

The answer is a **belief over** f — a posterior, a surrogate, a generative inverse.

SECOND CROSSING • LECTURE 7 → 8

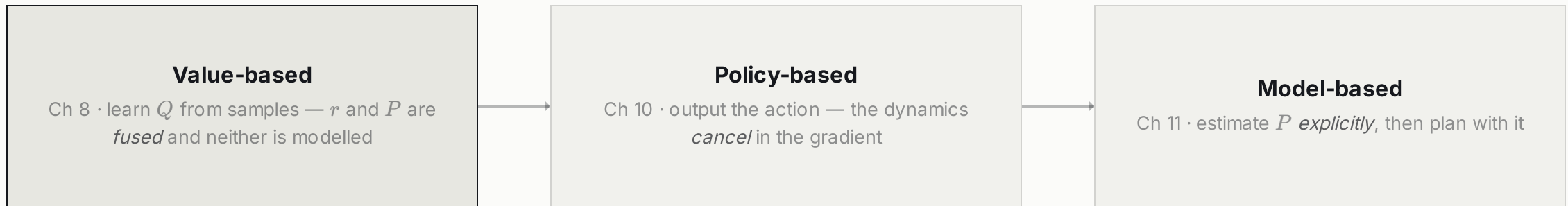
The dynamic world. Delete the model and **two** objects go missing at once:

$$Q^*(s, a) = \sum_{s'} P(s' \mid s, a) [R(s, a, s') + \gamma \max_{a'} Q^*(s', a')]$$

Reward r **and** transition P — the count doubles.

This is why RL is not merely **optimisation done with data**.

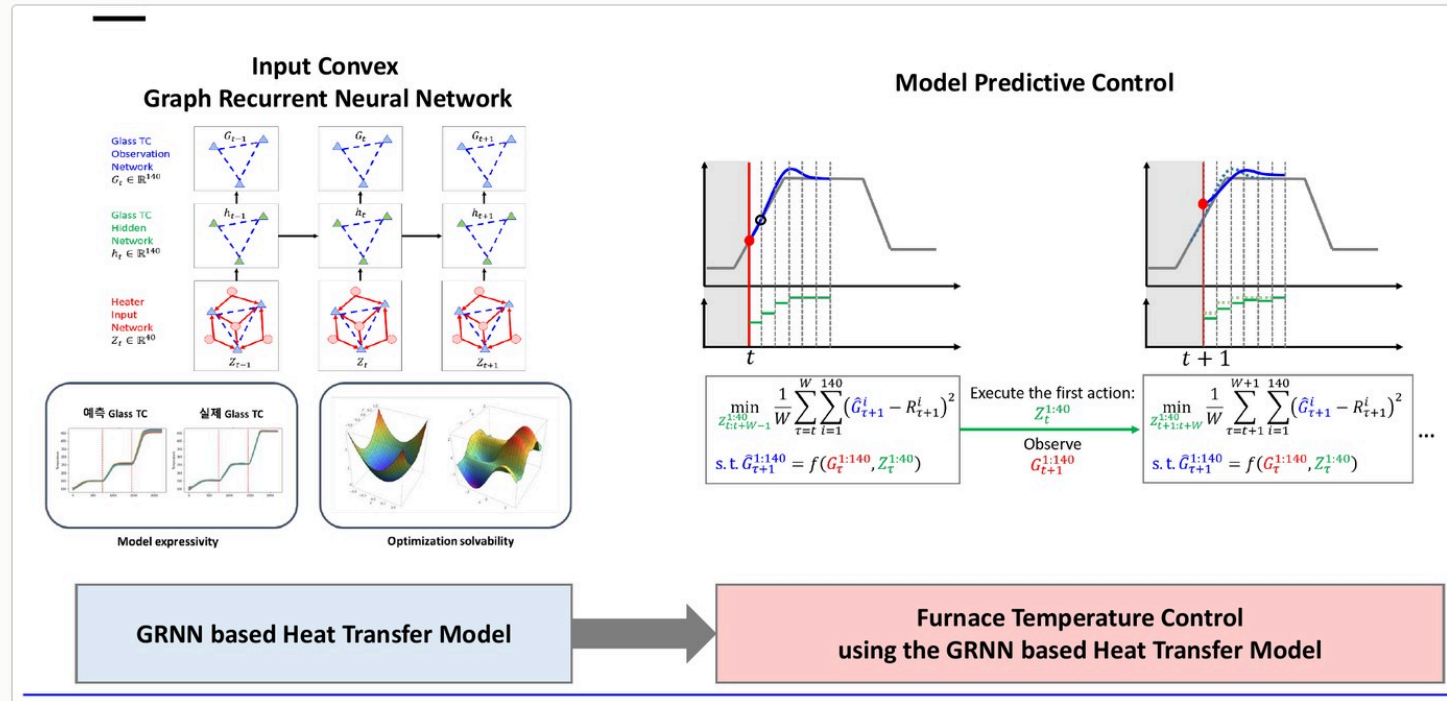
Three answers to the same doubling



Each answer pays a different price: fusing loses the ability to plan ahead; cancelling costs sample efficiency; estimating invites model bias. No answer dominates — which is why all three are taught.

Case C · Semiconductor furnace heat control

model-based RL, in production — the dynamics were learned, then planned with



An **input-convex graph recurrent network** learns the heat-transfer dynamics from operating logs; **model-predictive control** then optimises the work-set over a horizon and executes only the first action. **Convex in the control** is the design choice that makes the inner optimisation solvable at every step.

The balance the professor's own slide names: *model expressivity* against *optimisation solvability*. A more faithful dynamics model is worthless if the controller cannot optimise through it in the time between two ticks.

Check — follow the changing assumptions

Which sequence describes the main learning route of this course?

- ☐ A First remove the state, then the action, then the reward
- ☐ B First remove the data, then the model, then the objective
- ☒ C Learn about uncertain objectives, extend decisions through time, learn values/policies/models from experience, then work with a fixed log
- ☐ D First remove the dynamics, then the objective, then the horizon

This is a route through different assumptions, not a claim that every chapter removes exactly one assumption. Lectures 7 and 9 restore a known model to teach sequential decision making. Lectures 8, 10 and 11 use experience; Lecture 12 removes new interaction during training.

— 0.5

The single spine

Read the whole course as a slow stripping-away of what you were handed.

Each chapter removes one given

WHAT YOU WERE HANDED – AND WHEN IT IS TAKEN AWAY		Lecture 8
		
objective f what “good” means	not in play	
the right to query f may I evaluate a new x ?	not in play	
reward r the per-step objective	unknown – only samples	
dynamics P/f how the state moves	unknown – only samples	
sole decision maker nobody else optimises	handed to us	
VALUE-BASED RL		OBJECTS STILL TO BE LEARNED 2

Walk the lectures and watch the ledger empty. The counter on the right is the argument of the previous act: it reads **1** for the whole static half, and **2** from Lecture 8 onward.

The four systems, and the categories they belong to

Every method in this course was built against a real system. These four run through the lectures, and the numbers are the professor's own.

SYSTEM	CATEGORY	WHAT IS DECIDED	RESULT
Wind farm	② static, data-driven	where to place N turbines, in a shared wake	1.5% MAPE prediction; total power 7.5 → 20.0
Traffic signals	② static, data-driven (<i>meta</i>)	the green split at every intersection	408.5 → 395.5 waiting vehicles in Hangzhou; 920.1 → 859.7 in Manhattan
Semiconductor furnace	④ dynamic, data-driven	the heater work-set, every step	2.03 °C average error at a 100-step horizon
Combinatorial design	④ and beyond	a chip placement, a route, a schedule	search space $> 10^{90,000}$ states — beyond Go's 10^{360}

These are source-deck case studies, including simulations and reported applications. Their figures apply to the displayed setting; they are not performance guarantees on a new system — which is why the course spends its first six lectures on *how to state the problem* before it spends any on how to learn one.

The course, as a route through the map

-
- **Part I — the given world.** Ch 1: classical optimisation, the atom of every later method.
-
- **Part II — the uncertain world.** Ch 2 Bayesian statistics (belief as a distribution) → Ch 3 Bayesian networks (structured belief, plus decisions) → Ch 4 Bayesian optimisation (act on an unknown function).
-
- **Part III — design from data alone.** Ch 5–6 data-driven design optimisation: surrogate (forward) and generative (inverse).
-
- **Part IV — the world that unfolds.** Ch 7 MDP & DP → Ch 8 value-based RL || Ch 9 optimal control → Ch 10 policy-based RL → Ch 11 model-based RL.
-
- **Part V — when you cannot even try.** Ch 12 offline RL: the same dynamic, data-driven cell, with the right to interact withdrawn.
-

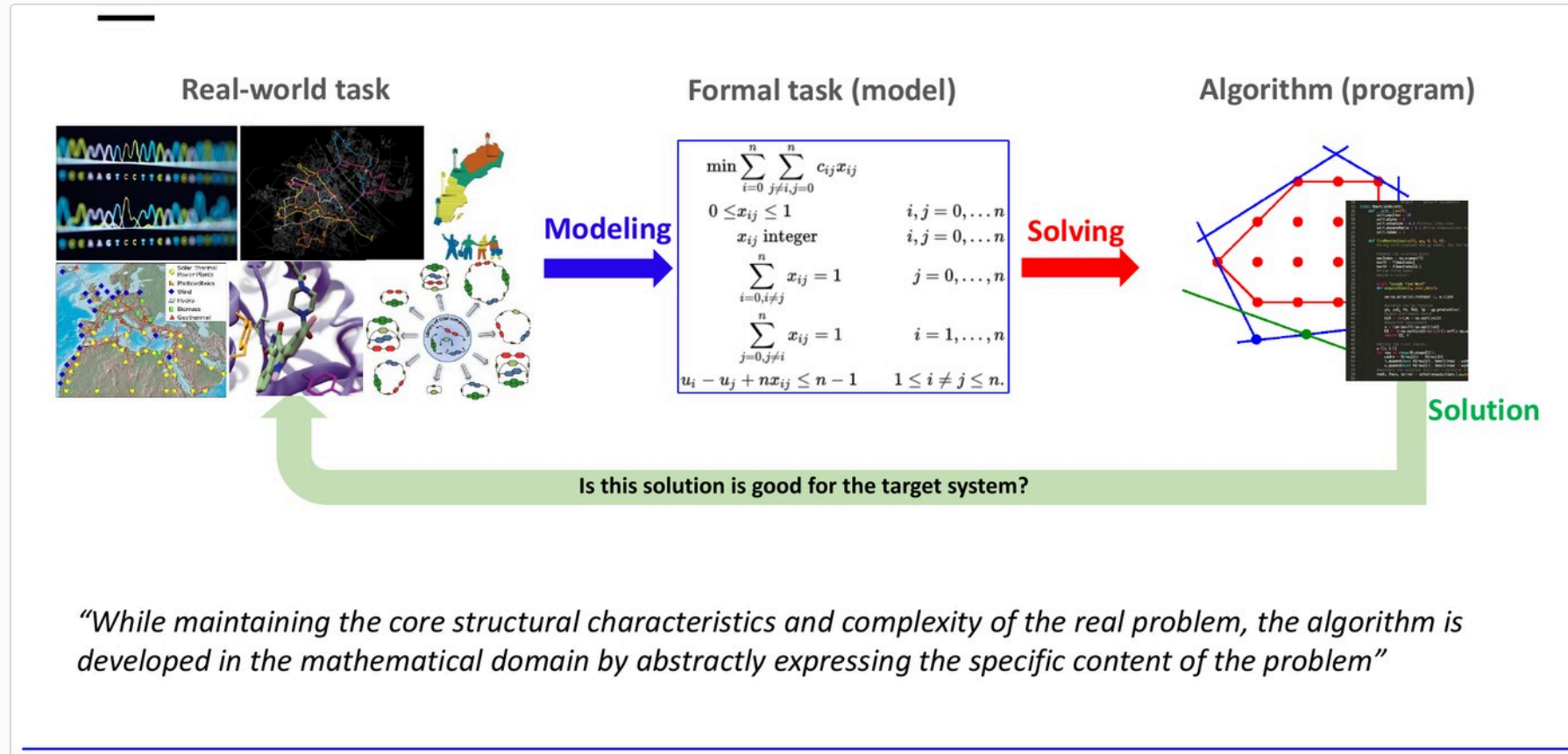
Appendix: a probability review — the toolbox underneath all of it.

What comes next

This final section is a **research and deployment outlook**. Connect each example to a decision, objective and constraint; detailed solver architectures are extensions to the core course map.

The bottleneck was never the solver

p. 69 of the source — the modelling loop, seen again with fresh eyes



The same three boxes as the opening slide. Solvers have become extraordinary; the arrow that has *not* improved is the first one. **Defining the problem mathematically is still the realm of advanced experts**, and there are not enough of them.

The paradigm shift

from writing a formulation to generating a solver

FORMULATION-BASED

A human expert reads the operation, writes $\min f$ s.t. $g \leq 0$, chooses a solver, tunes it.

Weeks per problem. Rewritten whenever the operation changes.

The expert is the bottleneck, and the expert does not scale.

GENERATIVE

A model is trained to *emit* a solution given an instance — **optimisation as inference**.

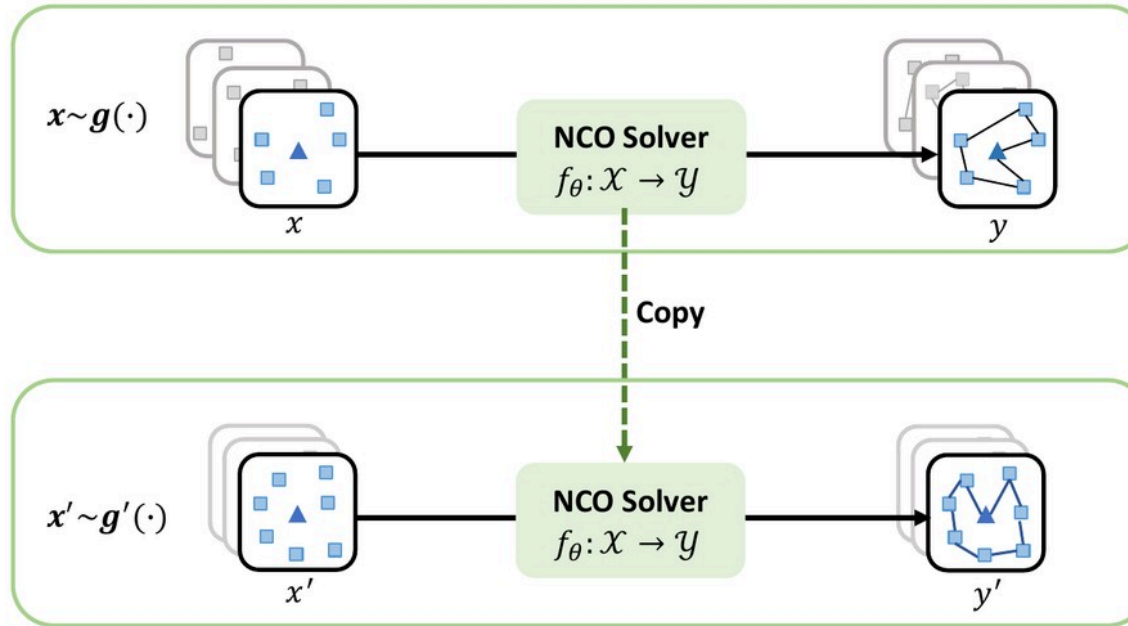
$$f_{\theta} : \mathcal{X} \rightarrow \mathcal{Y}$$

Milliseconds per instance, once trained. **The instance may be one the expert never saw.**

The source deck's own phrase for where this goes: **"ChatGPT for Optimization — generative AI for optimisation."**

Neural combinatorial optimisation, defined

- The objective of NCO is to learn a “Solver” to generate a good solution to a given target CO problem



NCO solver Design:

- Representation
 - Graph
 - Coordinate
- Decision making structure
 - Heuristics Based
 - Math Programming based
- Learning Methods
 - Imitation Learning
 - Reinforcement Learning

Performance Metrics:

- Generalization
- Optimality
- Speed

- Learning a NCO solver is solving **multi-task learning**:
 → **Meta-Learning** techniques is inherently need to be considered

Learn a **solver**, not a solution: $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$, trained over a *distribution* of instances $x \sim g(\cdot)$ with the aim of generalizing to $x' \sim g'(\cdot)$; a changed instance distribution must be tested. Because the training signal spans tasks, **learning an NCO solver is inherently multi-task learning** — which is why the meta-learning of Case B returns here.

What is learned in neural combinatorial optimisation?

original introduction PDF pp. 69–75 · separate the solver role from the training method

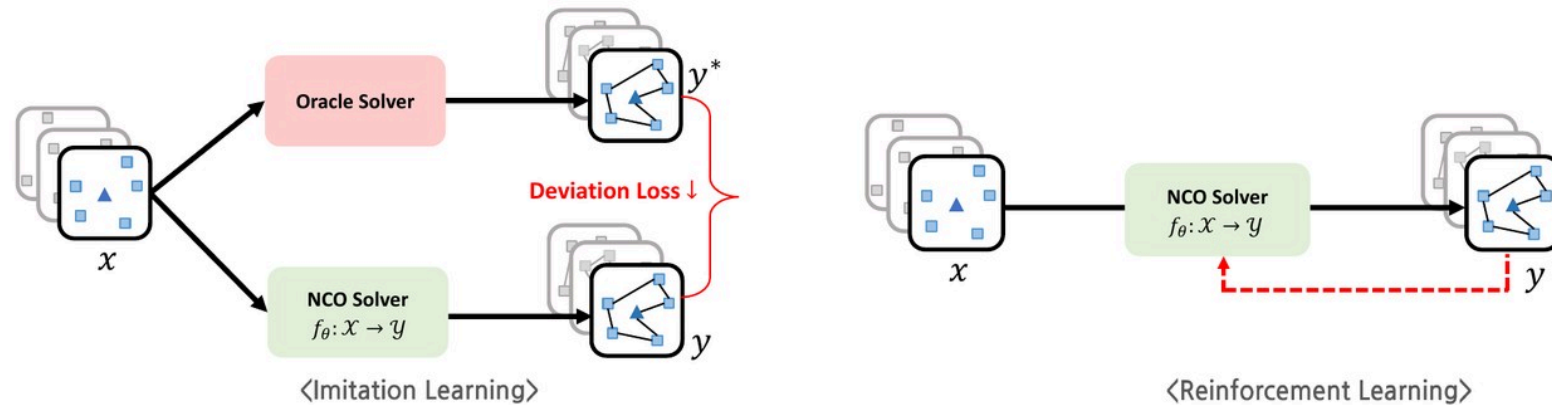
ROLE OF THE LEARNED COMPONENT	WHAT IT DOES	EXAMPLE
Construct a solution	choose the next element until a solution is complete	add the next customer to a route
Improve a solution	propose a change to an existing solution	exchange two route segments
Assist mathematical programming	guide a component of an exact solver	suggest branching or variable selection
Configure a solver	select settings for a problem instance	choose an algorithm or its parameters

Imitation learning and **reinforcement learning** are ways to train these components. They are a different classification from construction versus improvement.

Ask which decision the network makes inside the solver. A learned heuristic's speed or solution quality does not by itself prove global optimality.

Two ways to train it

Learning Methods: Reinforcement Learning vs. Imitation Learning



<Imitation Learning: optimization for minimizing deviation>

- Pros: easy to learn, fast to train
- Cons: requires the expert solutions, tends to over-fit

<Reinforcement Learning: optimization for maximizing performance>

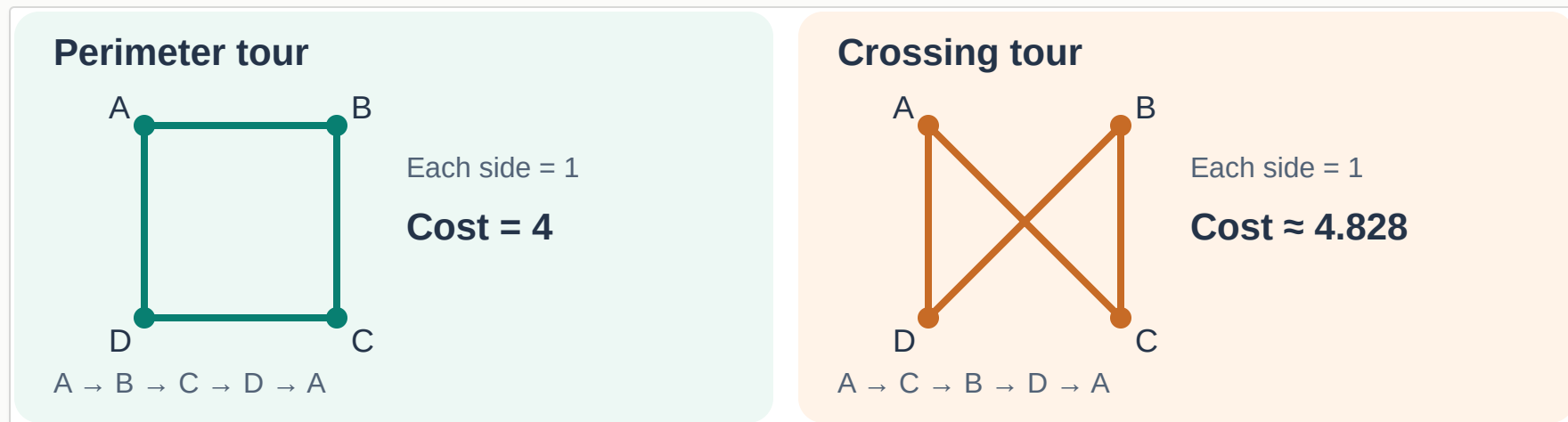
- Pros: labels (optimal solutions) is not required, inherent exploration → better generalization
- Cons: poor sample efficiency,

Imitation learning fits solver-provided solutions. **Reinforcement learning** uses reward or objective feedback. Their sample efficiency, solution quality, and generalization depend on the training setup; neither has a universal advantage. The choice is Lecture 8 and Lecture 10 arriving in a new domain.

A representative NCO paper — learn the next city

Kool, van Hoof & Welling · ICLR 2019 · a four-city teaching example

An attention model encodes the cities. A policy selects the next **unvisited** city, masks visited cities, and returns to the start after all cities have been visited.



The same four locations, two valid tours. These distances are an illustrative calculation, not a reported benchmark from the paper.

For unit-square cities, the perimeter tour costs **4**; the crossing tour costs $2 + 2\sqrt{2} \approx 4.828$. Training by REINFORCE compares a sampled tour with a greedy-rollout baseline and adjusts its probability using the cost difference.

The policy learns **how to construct** a solution; the objective still evaluates the resulting tour. Compare feasibility, tour length and computational budget with a conventional solver. [Attention, Learn to Solve Routing Problems!](#)

Your class project

Carry an industrial decision from its context to a tested prototype. Choose one appropriate core method and a baseline; explain the evidence for the resulting decision.

PART	WHAT YOU HAND IN	THE GATE IT MUST PASS
1 · Business context	the decision as it is made today, and what a better one is worth	a reader can state the scope in one paragraph
2 · Problem definition	objective, variables, hard and soft constraints, and the data contract	report feasibility checks and compare the KPI with the incumbent under the same conditions
3 · Product & delivery	the surface an operator would actually use	someone else can run it end to end

Every method in Lectures 1–12 is a candidate for part 2. The point of the gate is that **the choice of method is not the deliverable** — a validated decision is.

Check — two views of sequential decisions

Why does this course compare dynamic programming with optimal control?

- ☐ A They solve unrelated problems and share no mathematical tools
- ☐ B One can use values, while the other is forbidden to use them
- ☐ C Removing a model guarantees that a learned policy remains optimal
- ☒ D They provide complementary ways to reason about sequential decisions, which help explain value-based and policy-based learning

Both traditions use values, policies and Bellman reasoning. Lectures 7–10 organize the comparison as known-model methods followed by learning from experience. The columns are a teaching map, not disjoint historical categories.

— 07

Closing

One story, told in chapters — and a map to keep in view for the rest of the term.

One question — decide well under uncertainty —
asked along three criteria, answered by two
lineages.

Keep the decision, information and evaluation rule explicit as the assumptions change.

Let us begin — Lecture 1.

Start with a known objective and constraints. Each chapter will connect a previous skill to a new question, with a small calculation to try.

— APPENDIX

Examples, readings and derivations

Reference material for a second pass. The main teaching route ends before this section.

Reading guide — learn a solver, then recognise the real decision

one main idea to explain, one comparison, one application

ROLE	READ OR REVISIT	QUESTION TO ANSWER
Core	Kool, van Hoof & Welling, <i>Attention, Learn to Solve Routing Problems!</i> (ICLR 2019)	How can a policy construct a route one choice at a time?
Compare	A conventional routing heuristic and the NCO role table later in this lecture	What is learned: the solution, an improvement step, or a solver component?
Apply	The original wind-farm, traffic, furnace and routing cases	Which objective, constraints and evaluation budget define success?

First identify the decision. Read the routing paper for its construction loop and experiment design; detailed network architecture is optional. The engineering cases show where this pattern must meet physical and operational constraints.

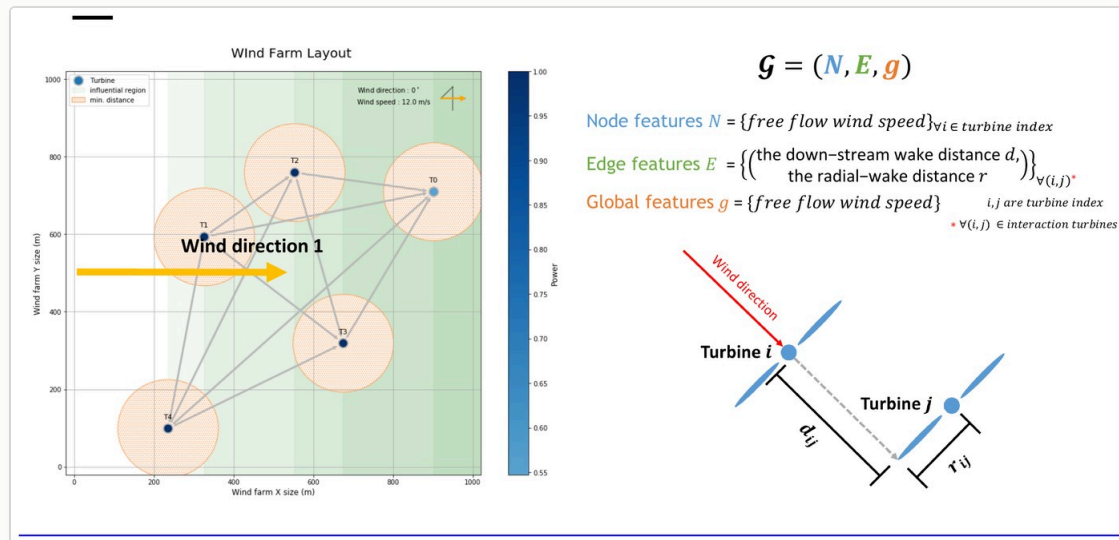
— EXTENSION

Case studies — engineering models and measured results

Read after completing the main route.

The farm is a graph, so the model should be one

THE REPRESENTATION

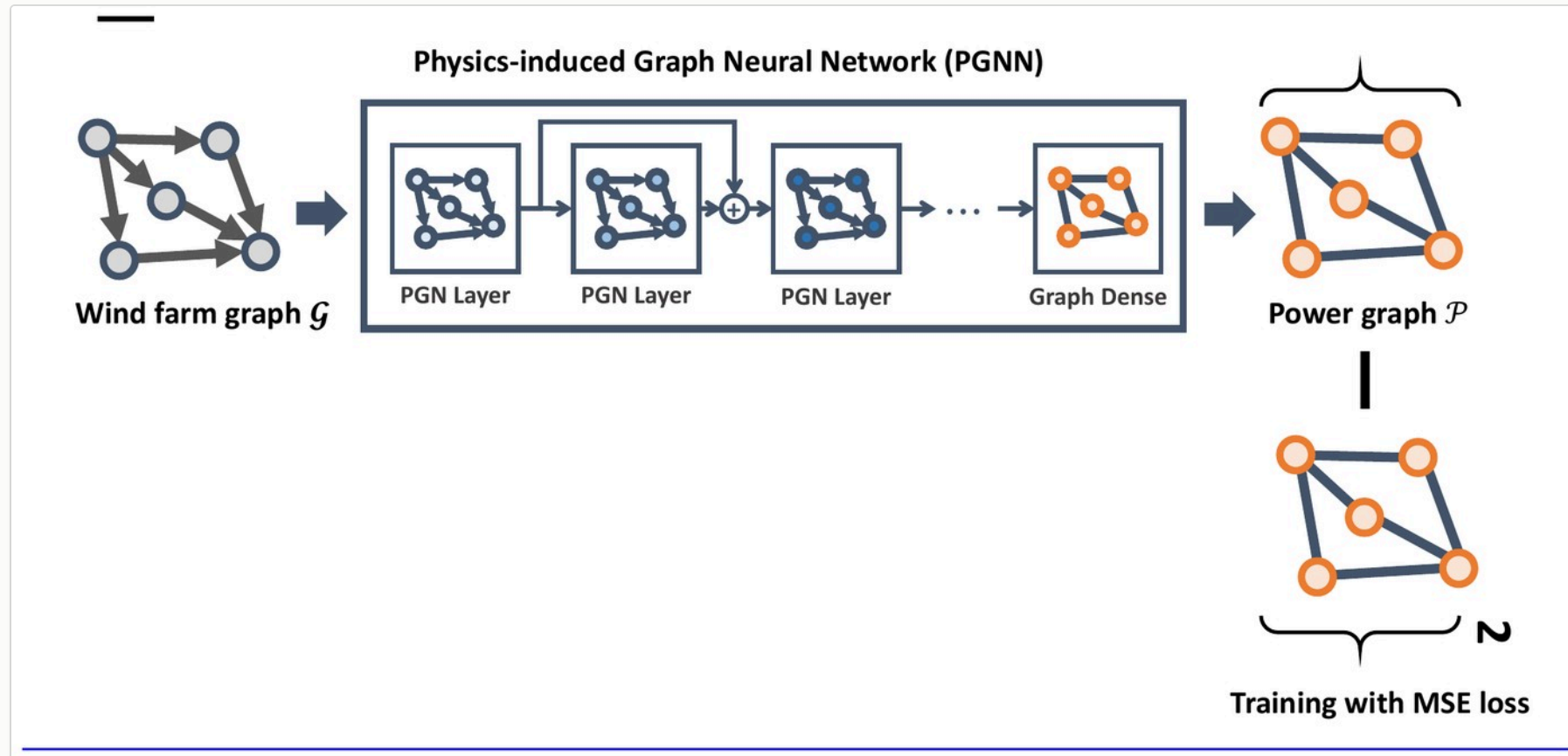


WHY A GRAPH NETWORK

Nodes are turbines and carry free-flow wind speed; edges carry the **down-stream wake distance d** and **radial distance r** ; the global feature is the wind itself.

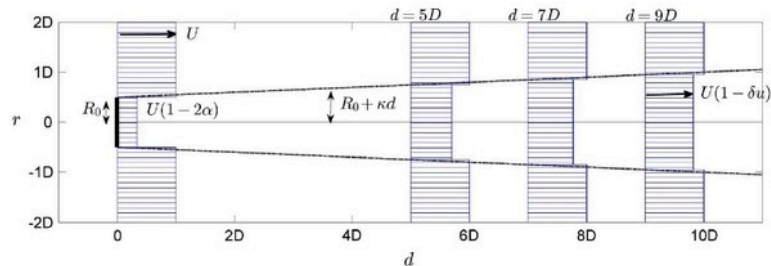
An appropriate graph readout is permutation-invariant, while node outputs are equivariant. It can accept different graph sizes; accurate transfer from five to twenty turbines still needs validation.

Physics-induced graph network — do not learn what you already know

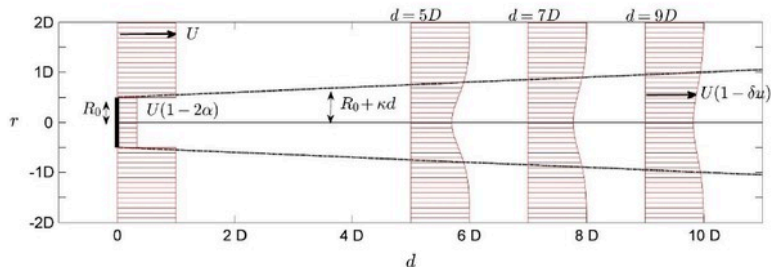


Four trainable modules and three aggregators, stacked. The edge update is where the physics enters: $\mathbf{e}'_{ij} = f_w(\mathbf{e}_{ij}) \times f_e(\cdot)$, with f_w a **wake weight** rather than a free function.

Let the network learn α, κ, R_0



(a) Park wake model (Jensen, 1983)



(b) Continuous wake model

$$\delta u(d, r) = 2\alpha \left(\frac{R_0}{R_0 + \kappa d} \right)^2 \exp \left(- \left(\frac{r}{R_0 + \kappa d} \right)^2 \right)$$

$\delta u(d, r)$ indicates
 'How much the down stream turbine is affected
 Due to the upstream turbines'
 → Weighting Factor w !

However, they tuned the parameters α, κ to the observed data

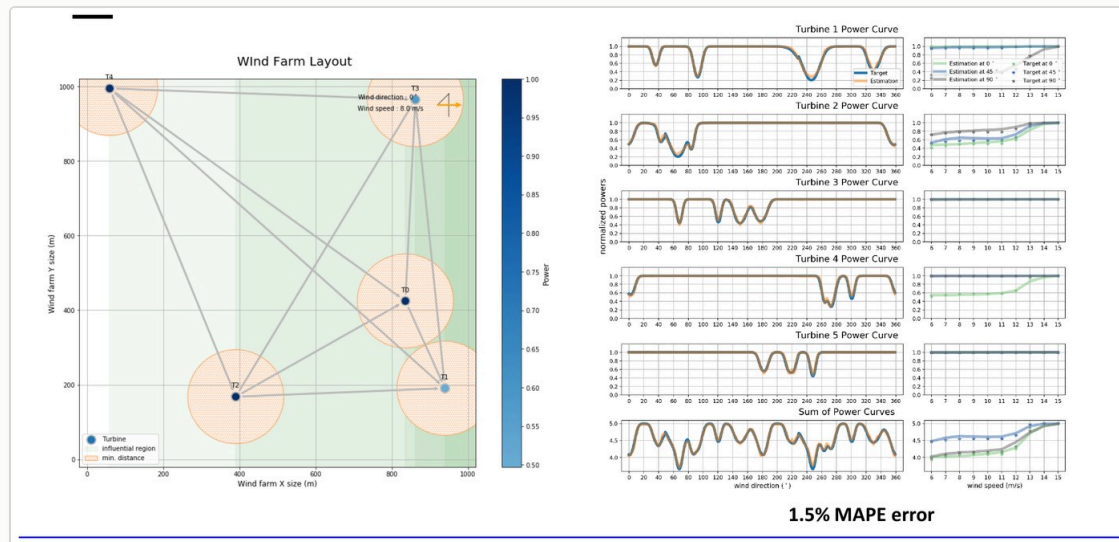
Let neural network learn α, κ, R_0 !

Figure source <Cooperative wind turbine control for maximizing wind farm power using sequential convex programming by Jinkyoo Park, Kincho H. Law >

The classical wake model gives the deficit $\delta u(d, r)$ in closed form, but its constants were **hand-tuned to observed data**. Keep the functional form, and learn the constants. This is the whole idea of an inductive bias, in one figure.

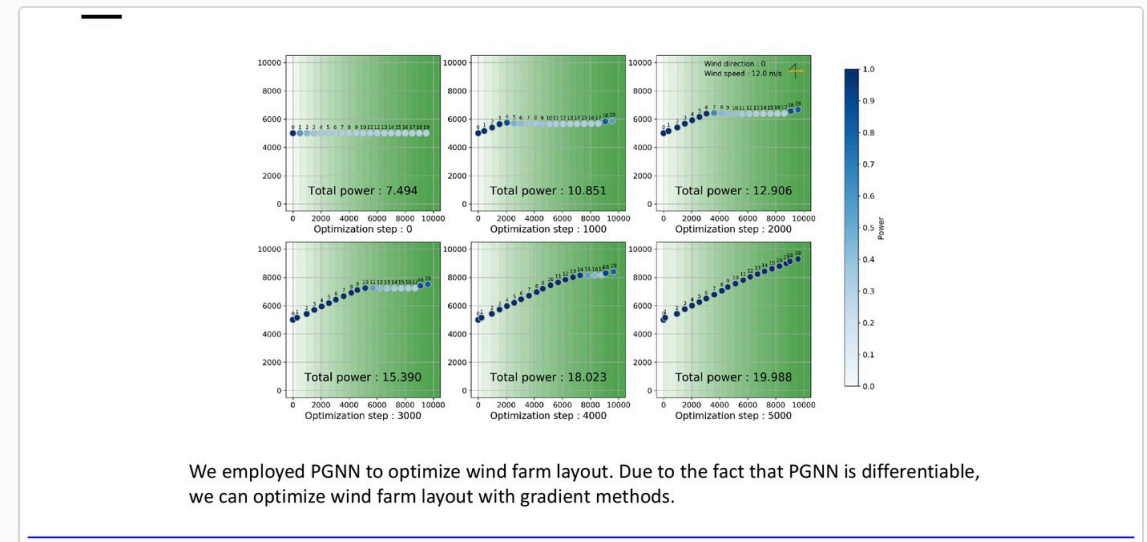
Does it work — and then, does it decide?

PREDICTION



1.5% MAPE on farm power.

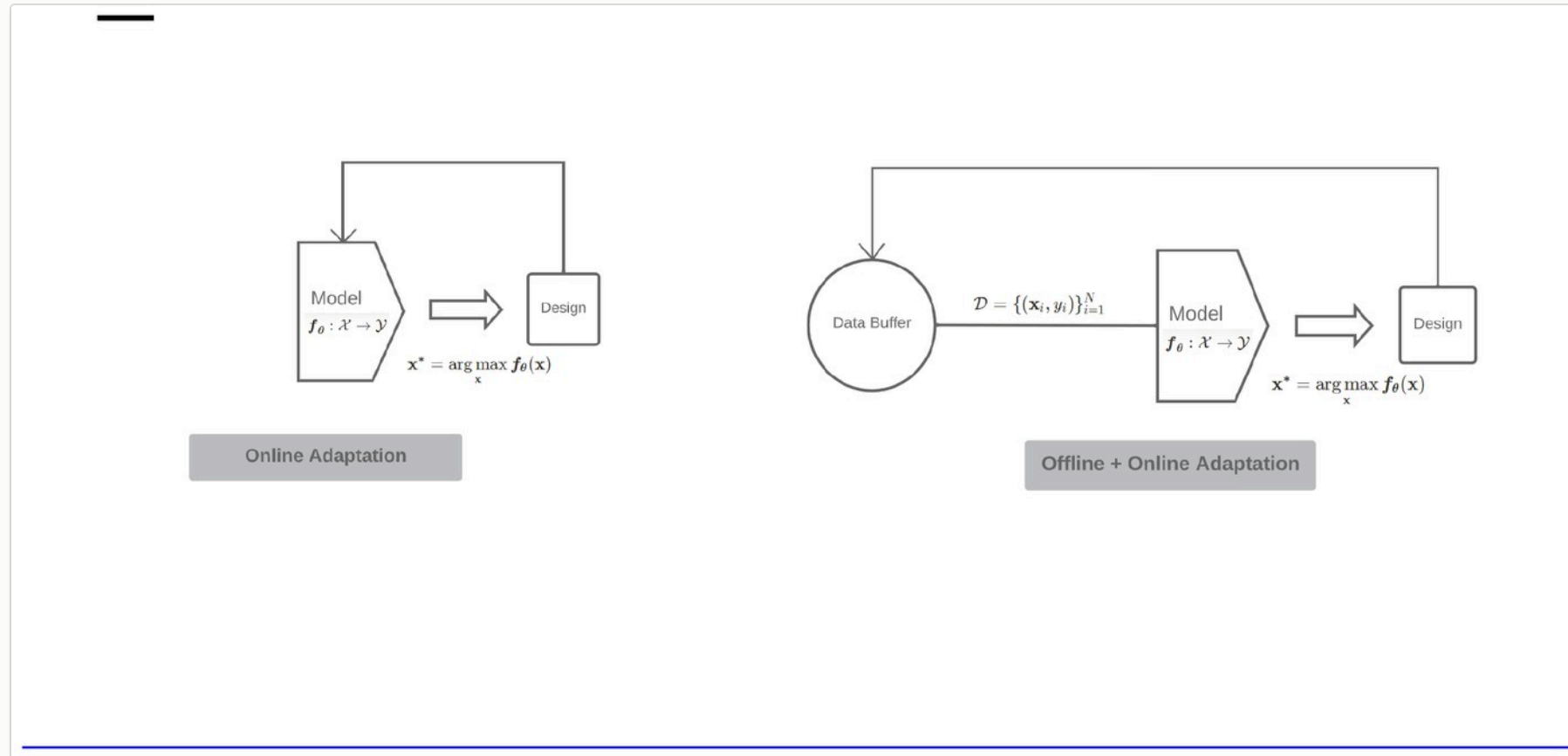
DECISION



Total power 7.5 → 20.0 over the optimisation, by gradient — because the surrogate is differentiable.

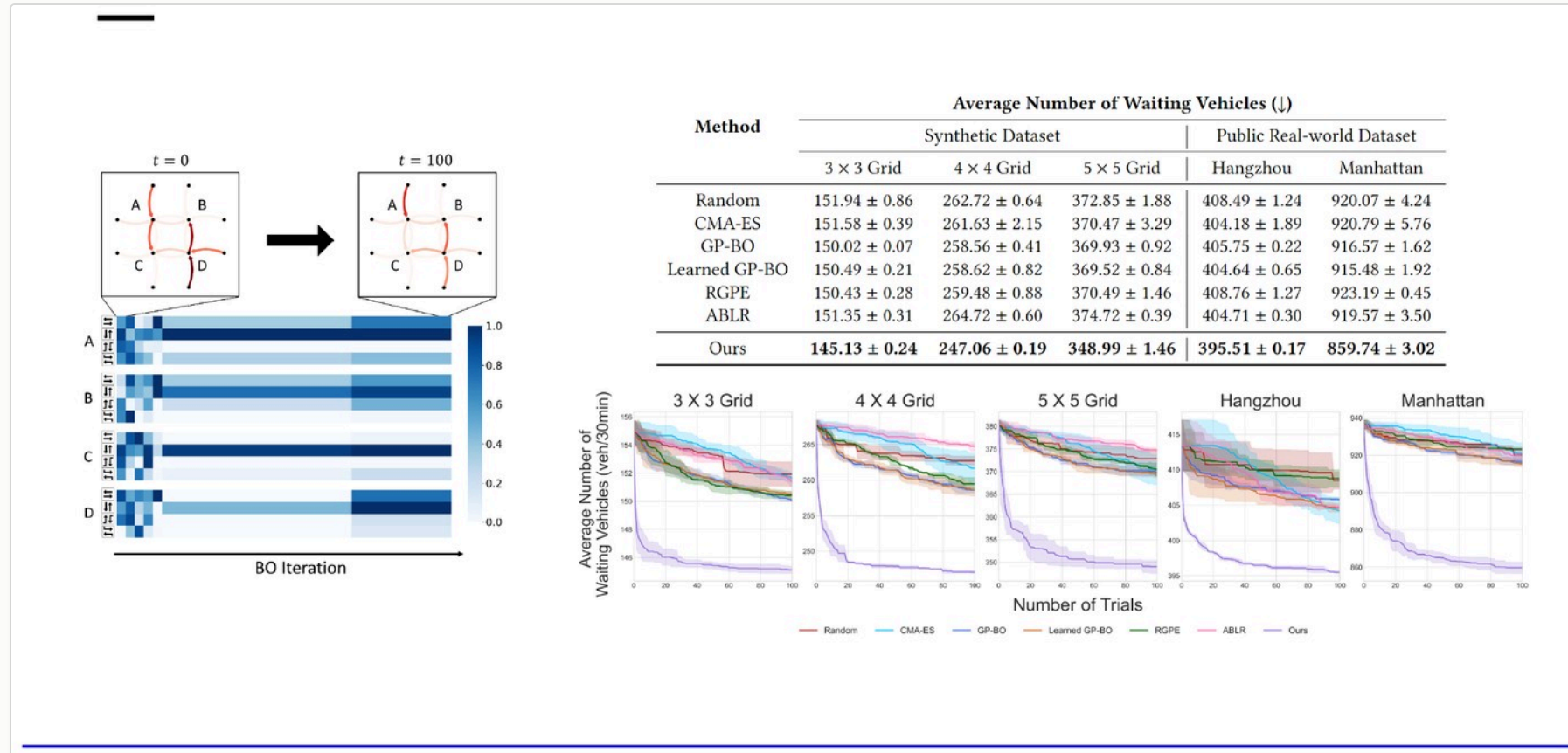
This is Lecture 5's thesis arriving early: a surrogate is worth building when it turns a problem you could only *simulate* into one you can *differentiate*. The warning that comes with it — that optimization can select optimistic surrogate errors — is Lecture 5's other half.

Why **meta** Bayesian optimisation



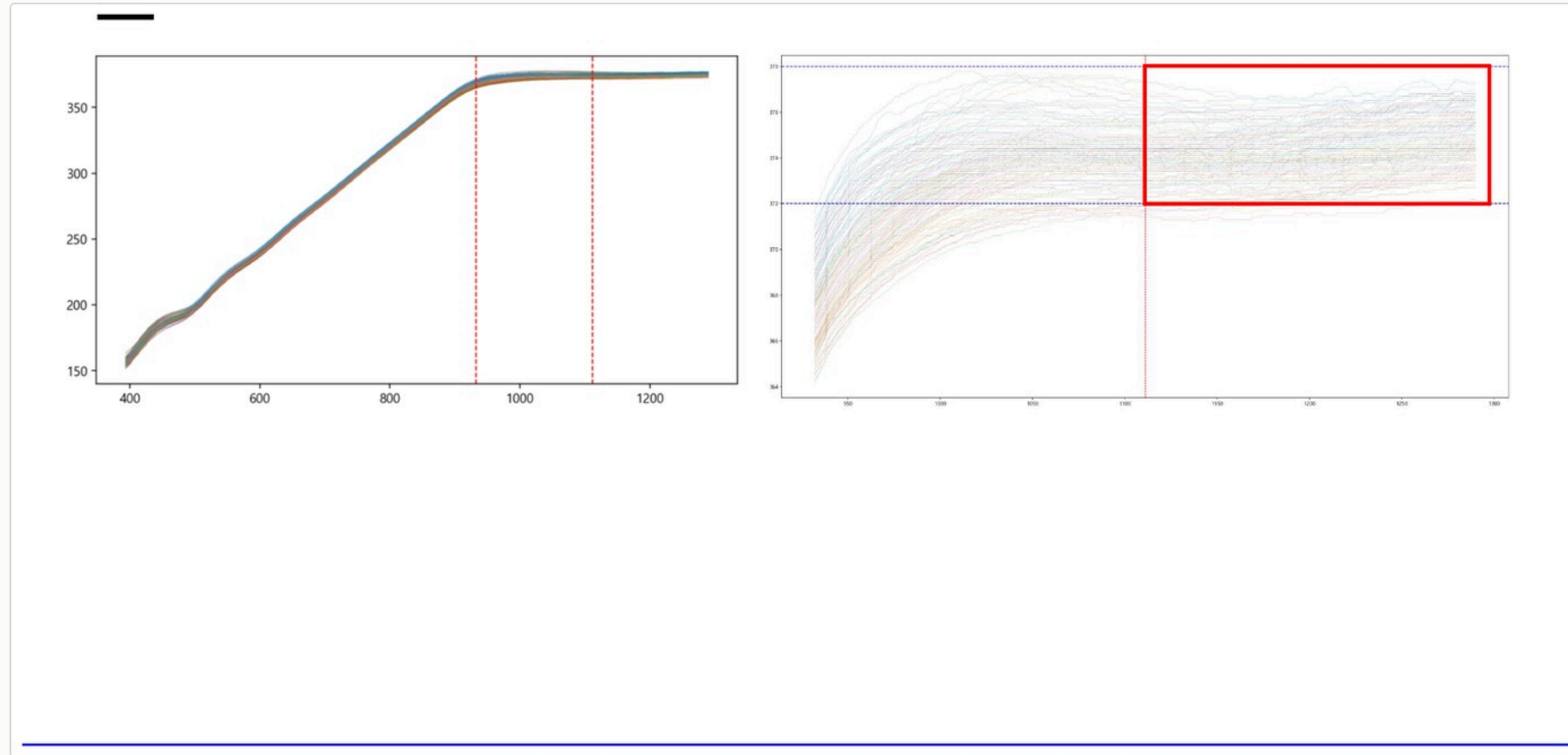
Ordinary BO can use an informative prior, but does not automatically transfer experience across intersections. **Meta-BO pre-trains f_θ on a buffer of past tasks**, then adapts online — so the first trial on a new junction is already informed.

The result



Average number of waiting vehicles. Ours is best on every column, and the gain is largest where trials are scarcest — **395.5 vs 408.5 in Hangzhou, 859.7 vs 920.1 in Manhattan.**

Case C • the result

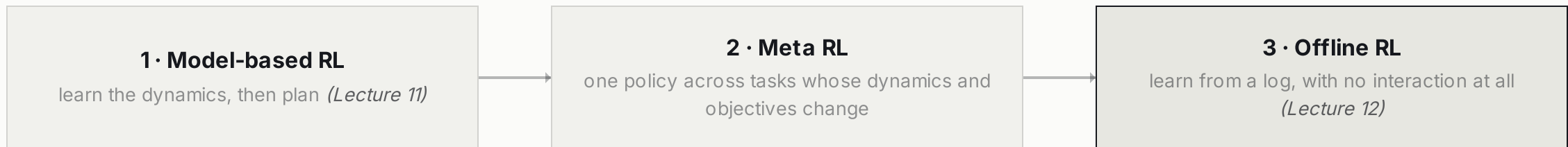


2.03 °C average error at a 100-step prediction horizon, and stable over 900 steps — enough that the overshoot region, the part an operator actually worries about, is predicted rather than merely tracked.

Extension toward practical RL

pp. 59 and 66 of the source — the professor's own roadmap

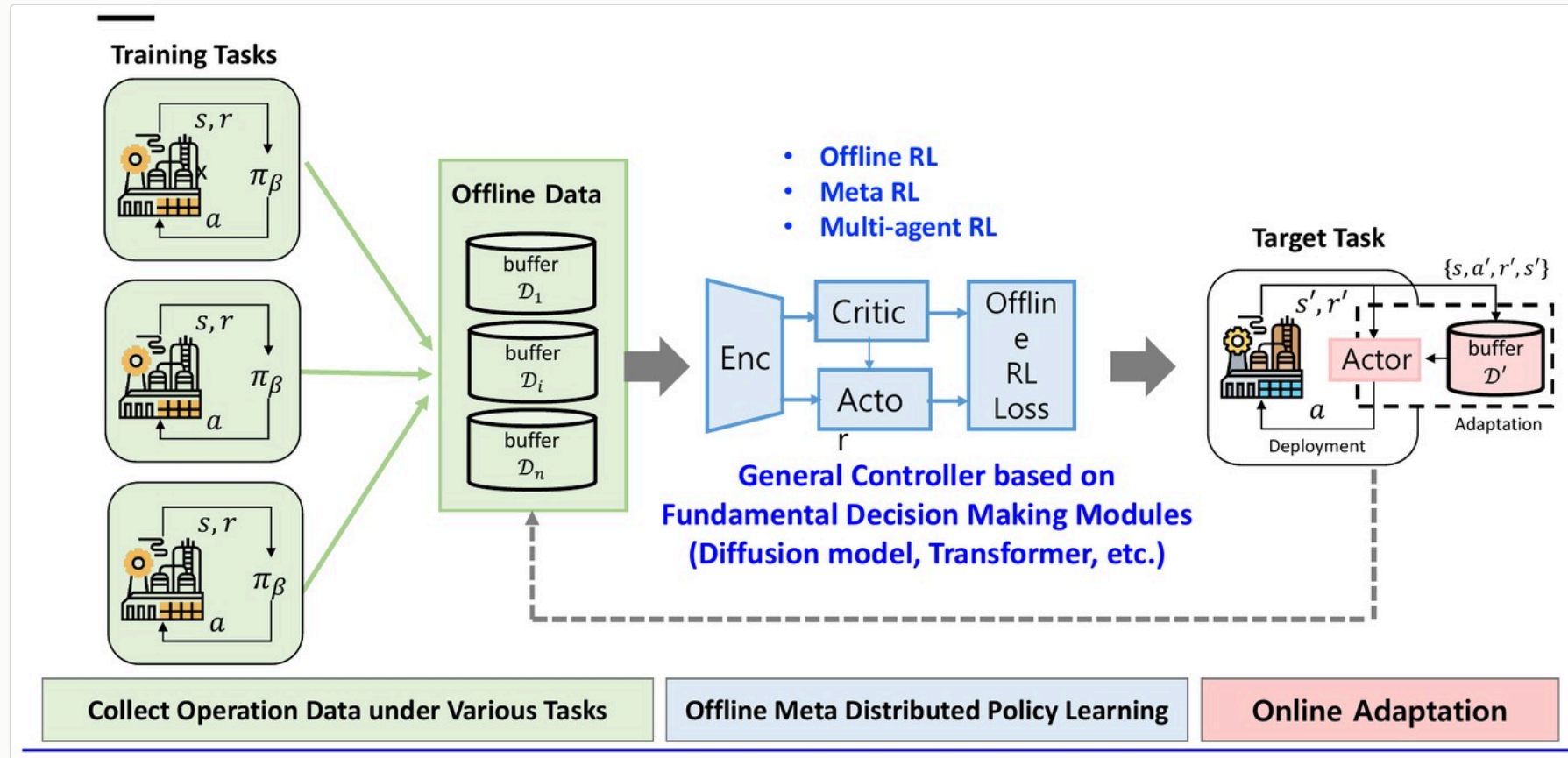
Many benchmark RL experiments allow extensive simulator interaction. The MDP formalism itself does not require a simulator or unlimited queries. Three extensions close the gap to a real plant, and this course takes two of them.



WHY OFFLINE RL — IN THE SOURCE DECK'S OWN WORDS

"Industrial systems typically do not have simulators, and it can be prohibitive to learn a policy by directly interacting with the real system. **Deriving a policy using previously collected data (offline data) is preferable.**"

Case D · Offline meta policy learning, then online adaptation



Collect operation data across many tasks; train a general controller offline — the encoder, the critic and the actor share a buffer per task; then adapt on the target task with **a handful of interactions, not a million**. Offline pretraining connects to Lecture 12. The later online adaptation is an extension beyond its fixed-data learning setting.

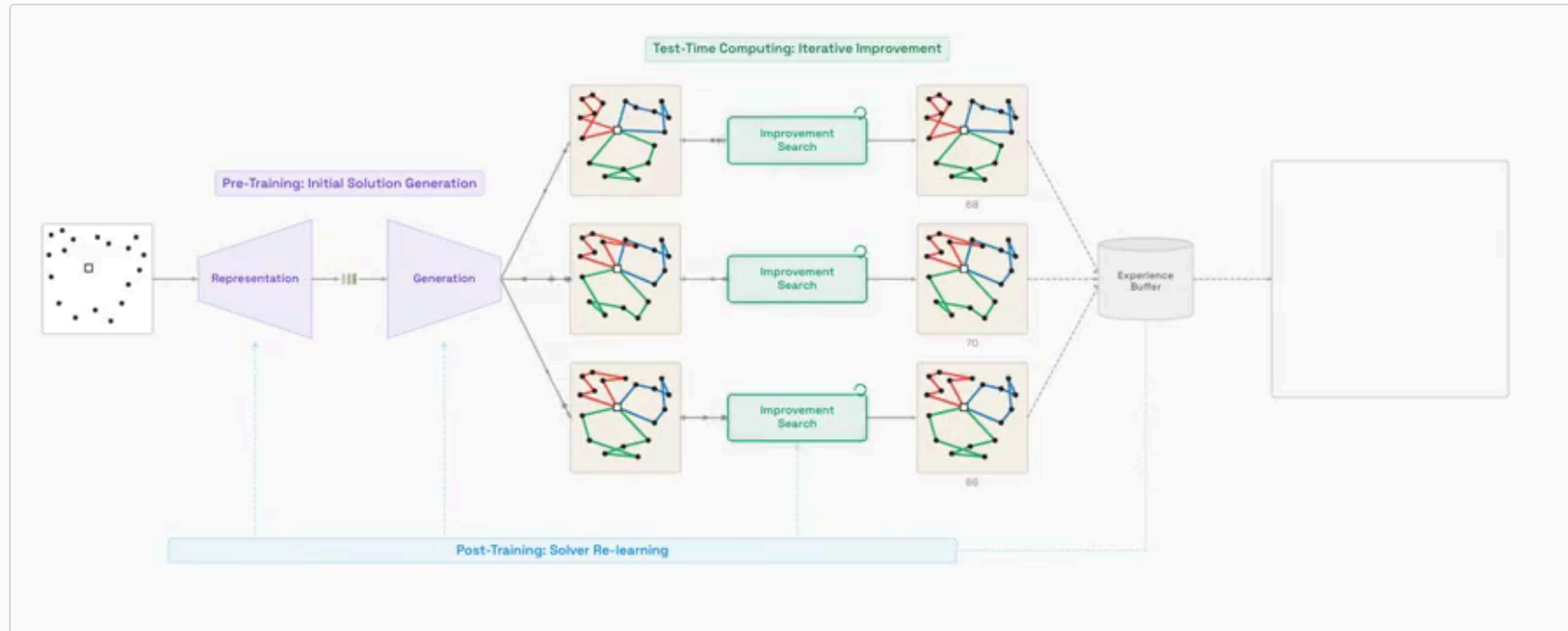
— EXTENSION

Research outlook — systems and applications

Read after completing the main route.

One engine, three domains — ① Vehicle routing

Case 1 of 3 · in production — generation, then test-time improvement search

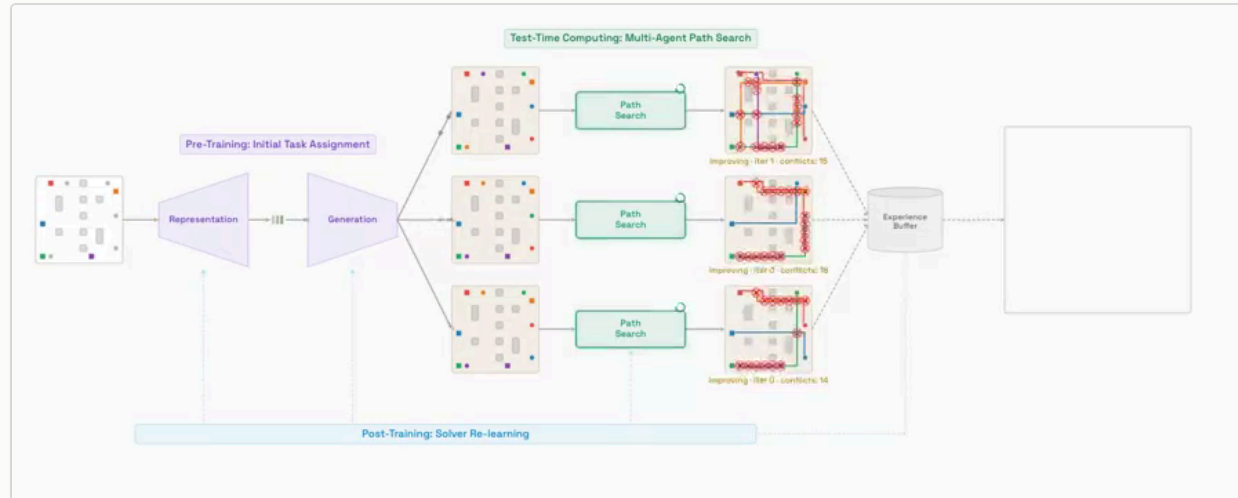


Pre-training generates several complete tours at once; **test-time search** improves each one and reports its length (62 · 64 · 61); the runs land in an **experience buffer** that **post-training** uses to re-learn the solver. **Nothing here is a hand-written heuristic** — the search operator is itself learned.

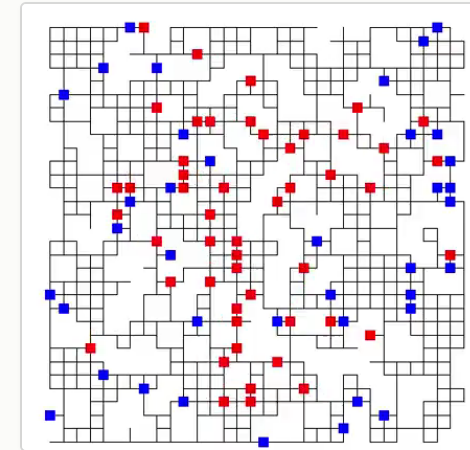
② Robot fleet — multi-agent path finding

Case 2 of 3 · same engine, new domain — generative assignment, then parallel plan improvement

THE SAME PIPELINE, RETARGETED



SILAB'S OWN RUN



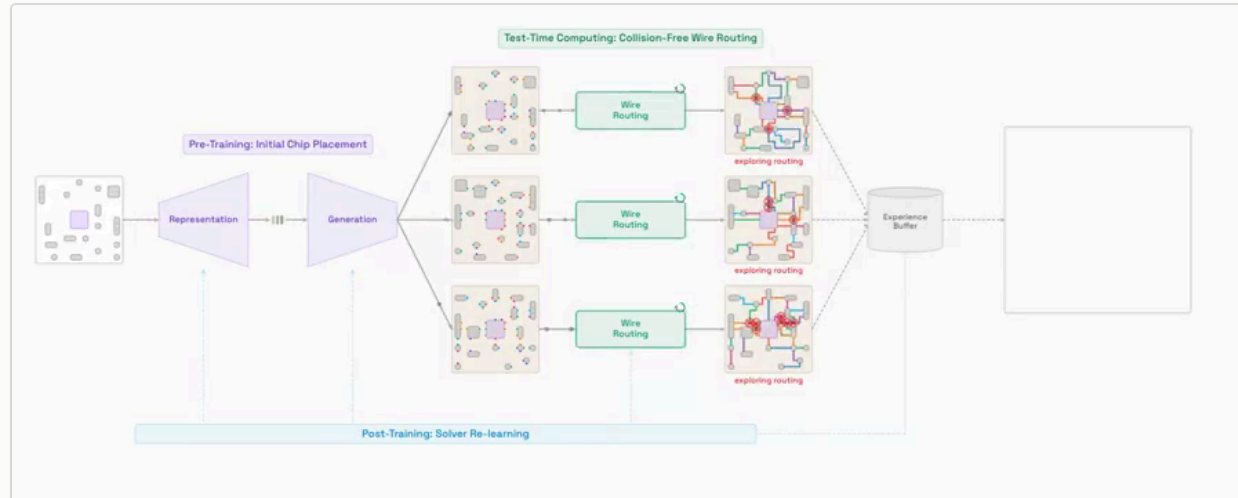
Agents blue, tasks red.

Generation now emits a **task assignment**; test-time search is a **path search** that reports the conflict count it is driving down (*iter 4 · conflicts: 12*). **The plan is not an answer until the conflicts reach zero** — which is a validator, not an objective.

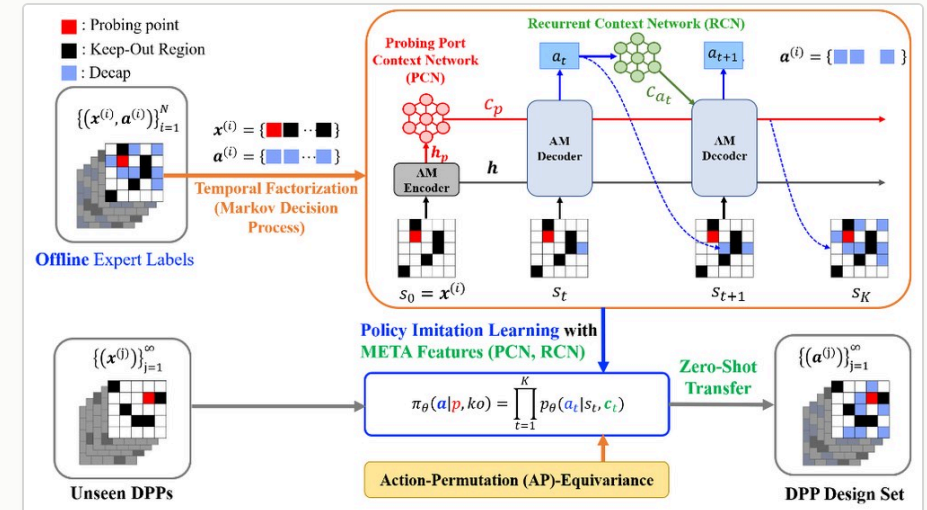
③ Chip placement and routing

Case 3 of 3 · beyond logistics — the same relational search engine, applied to silicon

GENERATION, THEN COLLISION-FREE WIRE ROUTING



META-LEARNED PLACEMENT



Placement is *meta* design: the task changes with the higher-level design flow, so what must be learned is a solver that adapts, not a placement that is fixed.

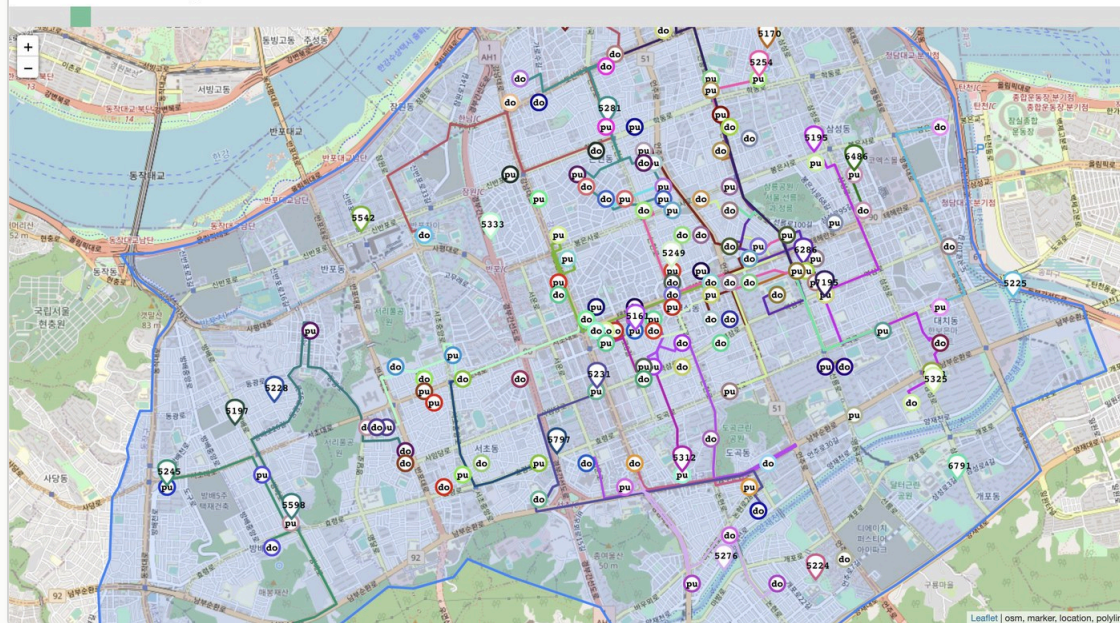
(NeurIPS'23)

The same engine, compared with a solver baseline

delivery routing, in paid production

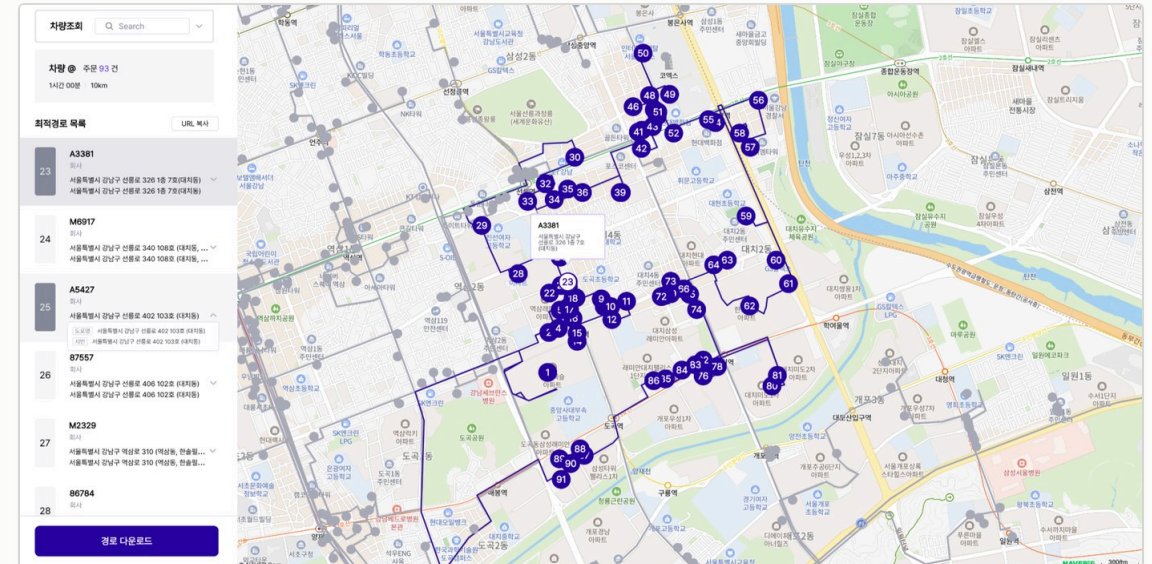
PACKAGE DELIVERY • GANGNAM

604 seconds passed



Distance **25.32** → **21.98 km**, delivery time **18** → **14 min** against Google OR-Tools.

FOOD & PHARMACY DELIVERY

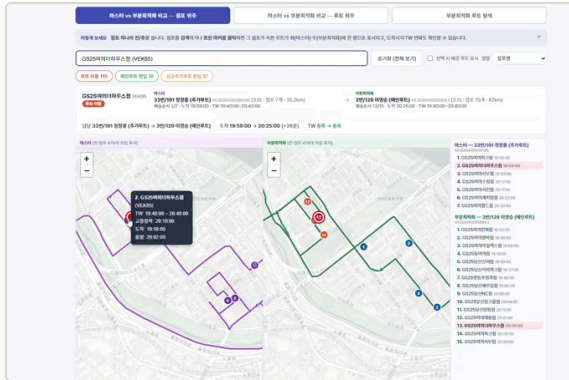


200 couriers, ~25,000 pharmacies. Total distance **873.5** → **695.1 km**; longest single route **67.4** → **37.4 km**.

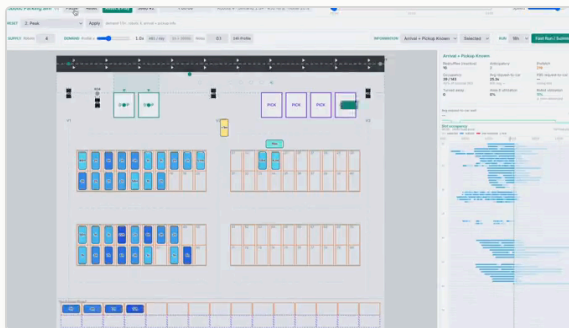
Running in the field, across domains

one engine, wrapped per domain — these are operating systems, not demonstrations

DELIVERY ROUTING



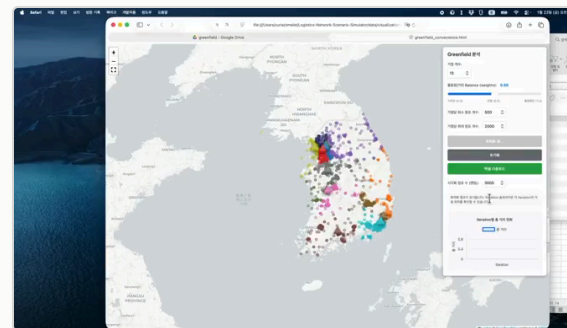
PARKING ALLOCATION



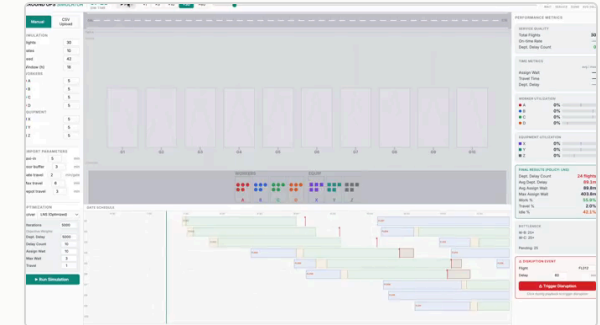
VESSEL STOWAGE



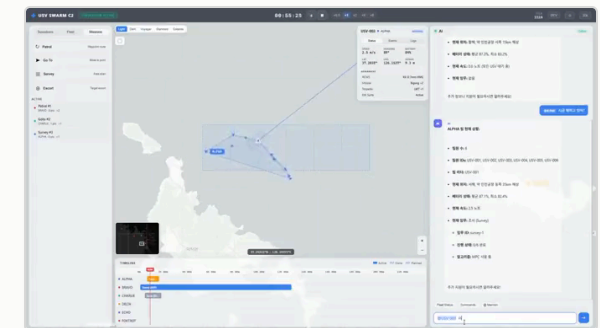
LOGISTICS NETWORK



AIRPORT OPERATIONS



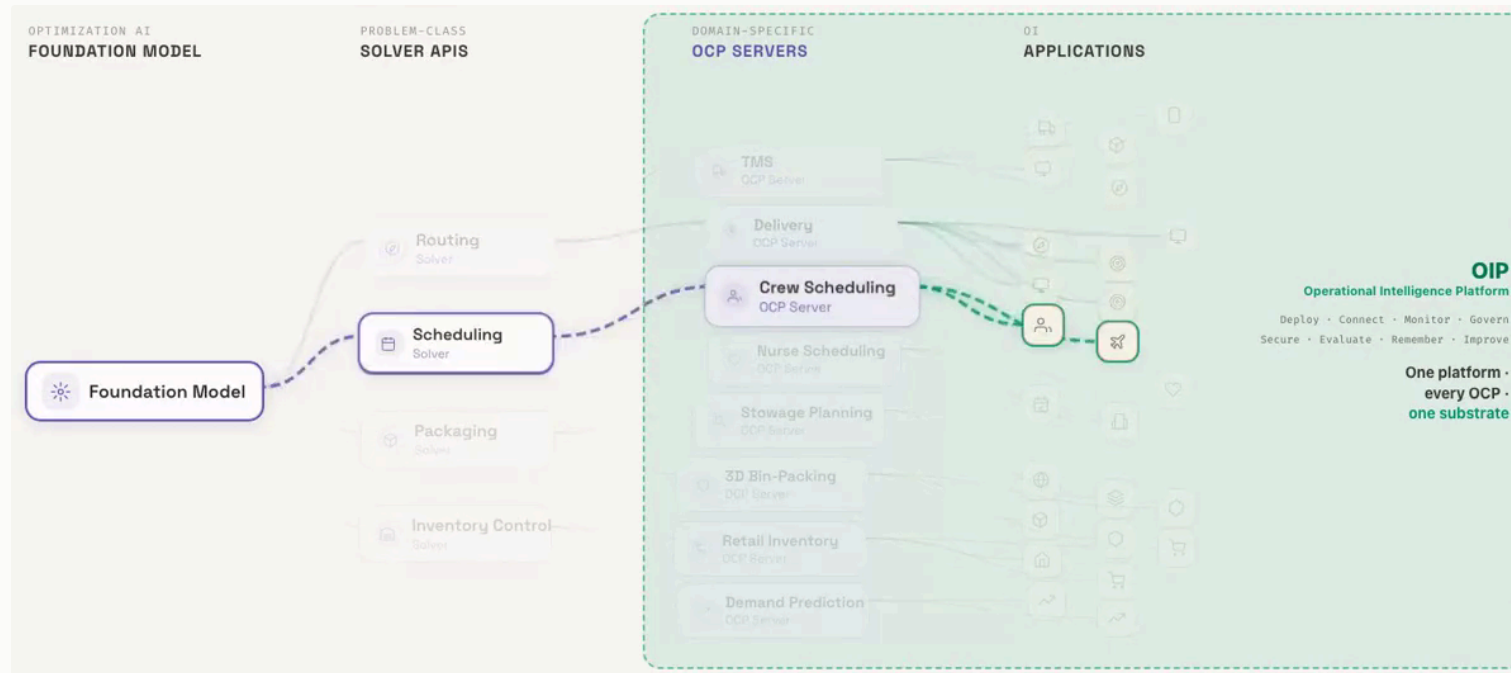
DEFENSE LOGISTICS



Each is the *same* combinatorial engine behind a different domain grammar and a different operator screen. **What is reused is the solver; what is rebuilt is the contract with the operation.**

The architecture this is heading toward

foundation model → solver → OCP server → agent

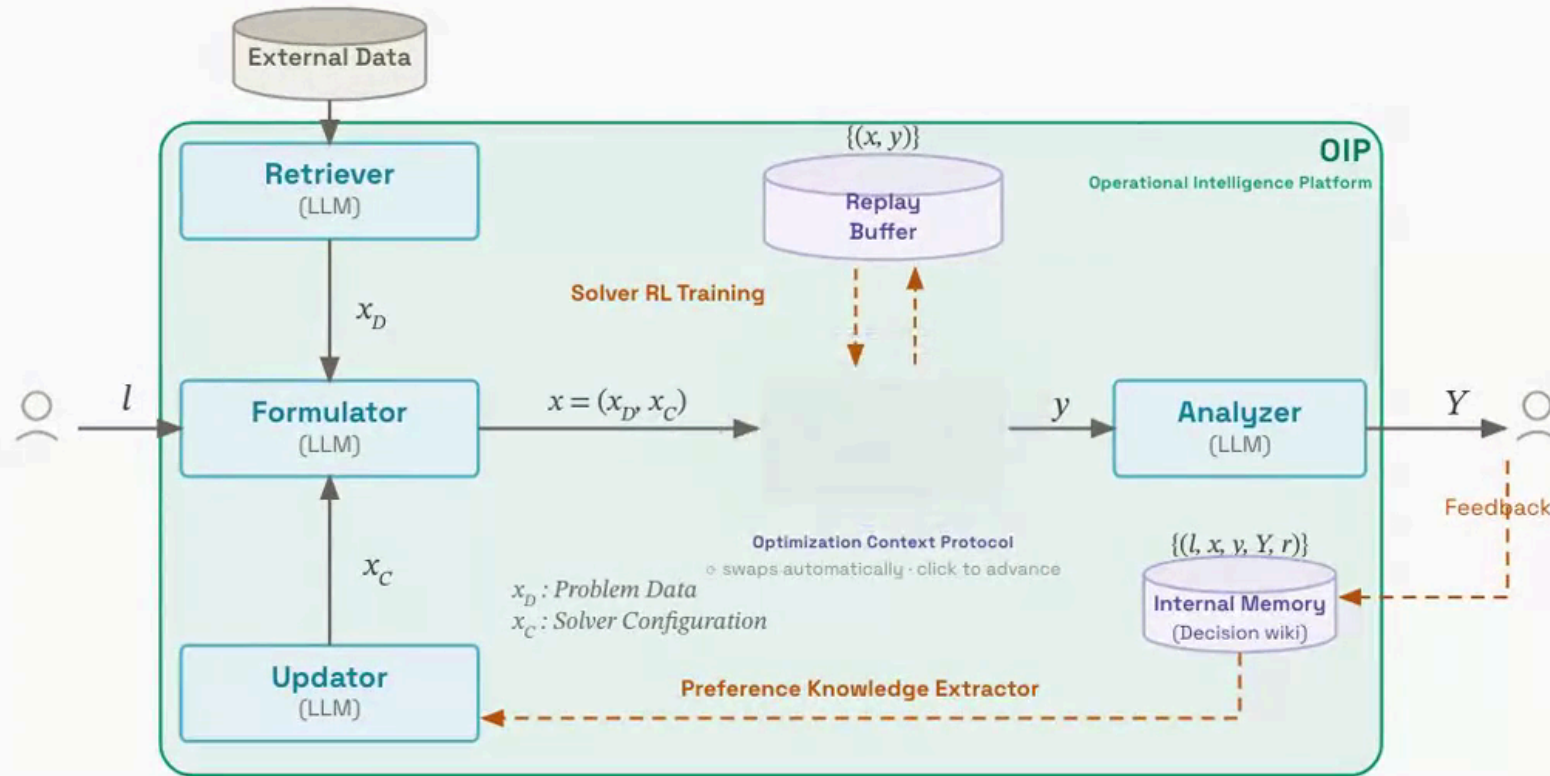


One **foundation model** feeds a small set of **problem-class solvers**; each solver is wrapped, per domain, into an **OCP server**; and the applications a field engineer opens are assembled from those. Watch the paths light in turn — one model, one solver, four different operations. **Every layer exists to make the one before it reusable.**

Each layer makes the one above reusable. The foundation model creates the intelligence, the **OCP server** (Optimization Context Protocol) makes it reusable, and the platform makes it operable — deploy, connect, monitor, govern, remember, improve.

The optimisation engine, talking to a language model

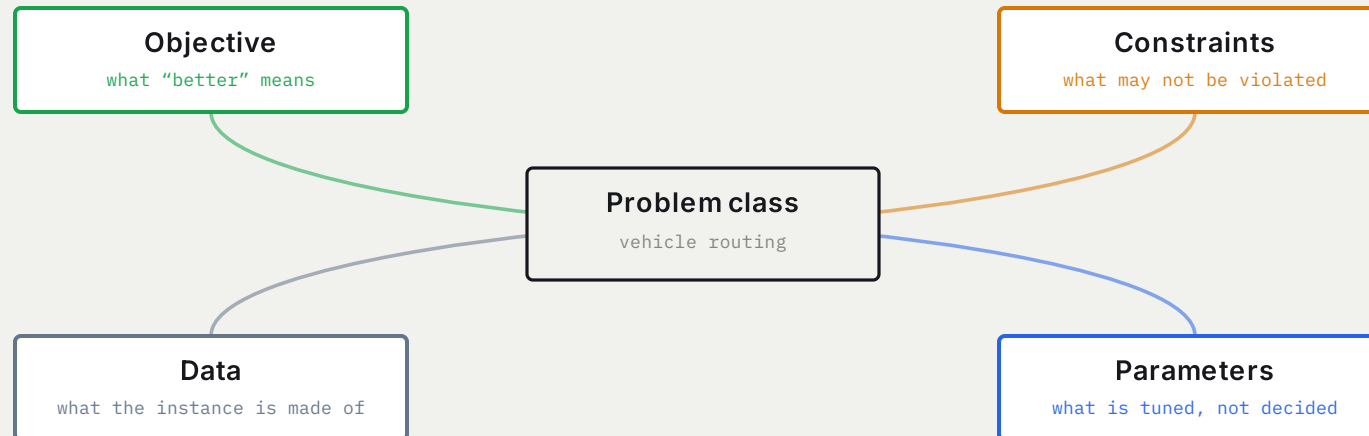
this is the structure that puts optimisation in a field engineer's hands



A request in words (l) is turned by the **Formulator** into an instance $x = (x_D, x_C)$ — problem data and solver configuration — which the **OCP server** solves; the **Analyzer** returns the answer in the engineer's language. **Language models do the reading, writing and explaining; the solver does the deciding.** Nothing here asks an LLM to optimise, and nothing asks the solver to understand English.

The knowledge that makes the translation possible

THE ONTOLOGY – FOUR THINGS A FORMULATION NEEDS TO KNOW

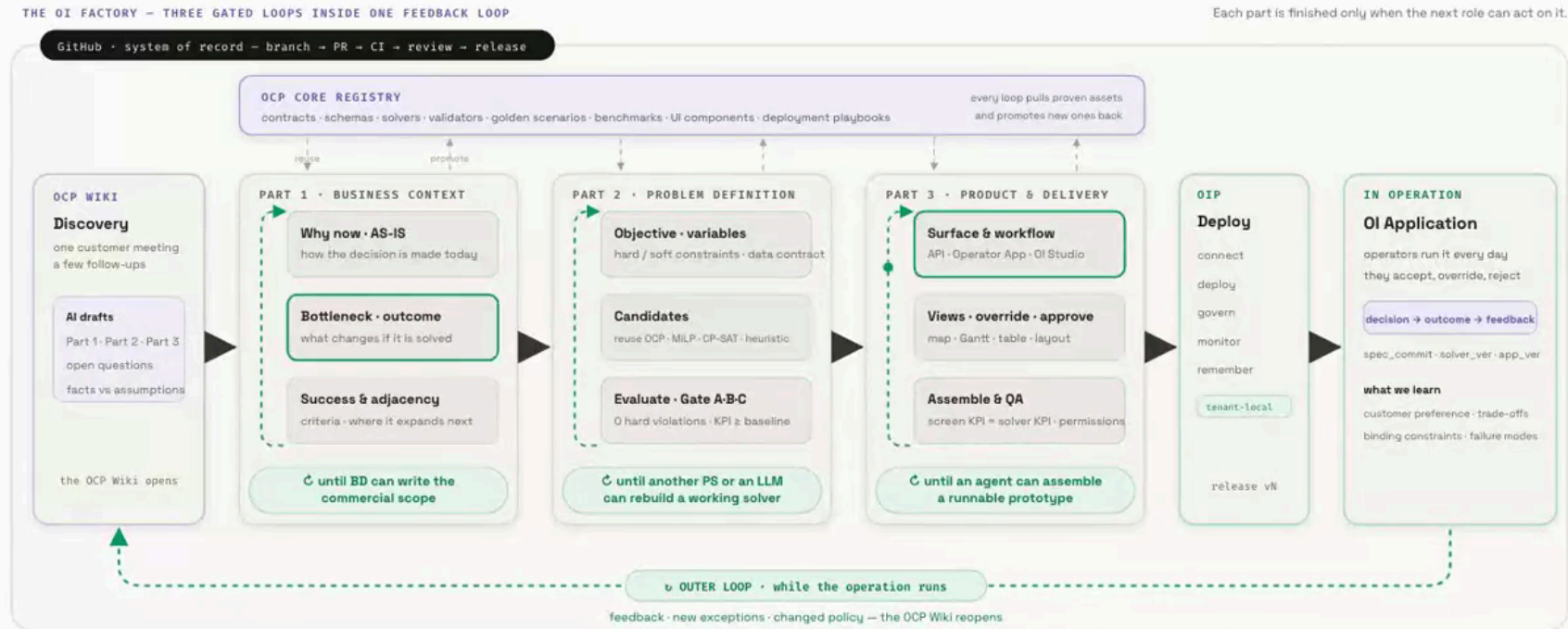


Supplied once per problem class and reused by every run. The formulator does not *invent* a model from a sentence — it **resolves** the sentence against this, which is why the same request twice gives the same formulation twice. Click a facet.

An ontology over **objective, constraints, data and parameters** is what turns a sentence into a structured formulation, while exposing missing information and ambiguities that still need resolution. Supply it once per problem class, and every later run inherits it.

The OI Factory

three gated loops inside one feedback loop — and the shape of your project



Three gated loops — business context, problem definition, product and delivery — each turning until the next role can act, all inside one outer loop that reopens when the operation talks back. **Your project is one pass through this.**

THE RULE THAT MAKES IT A FACTORY, NOT A PROJECT

Each part is finished only when **the next role can act on it**. Part 2 is done when another person — or a model — could rebuild a working solver from it alone.