MODULE 1 · SESSION 1

Architect a Multi-Model Product

How several specialised models can cooperate inside one product, each with a defined job and a clear handoff between them.

Theory session

Beginner

Running case: LumaForge Robotics

ORIENTATION

Where this session sits

Module 1 builds the foundations: architecture, data, and routing

MODULE 1
AI System Foundations

MODULE 2
Controlled AI Operations

MODULE 3
Strategic AI Operating Choices

SESSIONS IN THIS MODULE

1.1 Architect a Multi-Model Product — YOU ARE HERE
How specialised models cooperate inside one product

1.2 Establish Synthetic Data Trust
Traceable data, and knowing which evidence is real

1.3 Route Models Across Workloads
Choosing a model by task, quality, latency and cost

SESSION OUTCOMES

What this session explains

Designing a product where several models work together

01

One model rarely does everything well

Different jobs have different requirements for accuracy, speed, cost and safety.

02

Architecture means deciding the boundaries

Which component is responsible for what, and what passes between them.

03

The interface is the design

A clean handoff is what makes a system testable, debuggable and replaceable.

04

Four common patterns cover most products

Pipeline, router, ensemble and supervisor — each with a different shape of failure.

DEFINITION

What a multi-model product is

And what it is not

WORKING DEFINITION

A multi-model product is one in which two or more models each handle a defined part of the work, connected by explicit interfaces, so that each part can be measured, improved or replaced on its own.

1

Not just more models

Adding a second model without defining its responsibility makes a system harder to reason about, not easier.

2

Each part is separately testable

If a picking error occurs, you can tell whether the scene was misread or the motion was wrong.

3

Each part is separately replaceable

A better vision model can be swapped in without retraining the controller.

CONCEPT 1 · MODULARITY

Why splitting work into modules helps

The oldest idea in system design, and still the most useful

INDEPENDENT IMPROVEMENT	Each module can get better without the others changing.
LOCALISED FAILURE	When something goes wrong, the boundary tells you where to look.
PARALLEL WORK	Different people can build different modules at the same time.
OPTION VALUE	You can replace one supplier or model later without rebuilding the product.

Parnas, D. (1972). On the criteria to be used in decomposing systems into modules. Communications of the ACM, 15(12), 1053–1058.

THE RULE

Hide what changes

Put the things most likely to change inside a module, so the rest of the system does not depend on them.

THE COST

Boundaries are not free

Each handoff adds latency, a place for information to be lost, and something else to monitor.

THE WAREHOUSE EXAMPLE

Scene understanding and motion control

Two modules, two very different rates of change — which is exactly why the split was worth making.

CONCEPT 2 · PATTERNS

Four ways models are commonly connected

Recognising the pattern tells you where it will break

MODELS RUN TOGETHER ▲

ENSEMBLE

Several models answer the same question; their answers are combined. Better accuracy, higher cost.

SUPERVISOR

One model checks or corrects another's output before it is used.

PIPELINE

Each model does its stage and passes the result on. The warehouse VLM → VLA pattern.

ROUTER

One component chooses which model handles each request. Session 1.3.

ONE MODEL DECIDES FOR ANOTHER →

PIPELINE

Errors travel forward

A misread scene becomes a confident wrong action. Test each stage, not only the end result.

ENSEMBLE AND SUPERVISOR

You pay twice

Two inferences per request. Worth it when being wrong is expensive; wasteful when it is not.

CONCEPT 3 · INTERFACES

What has to be agreed at every boundary

Five things, written down before either side is built

What to define	Why it matters	Warehouse example
The shape of the message	Both sides must agree on fields and units.	Object positions, grasping order, placement targets.
What counts as valid	So bad input is rejected instead of acted on.	A placement outside the reachable workspace is refused, not attempted.
Confidence or uncertainty	The receiver needs to know how much to trust it.	A low-confidence scene reading triggers a second look rather than a grasp.
Timing expectations	A late answer can be worse than no answer.	A control command must arrive inside the robot's cycle time.
What happens on failure	Every interface needs a defined fallback.	No usable plan means stop safely, not guess.

> **A PRACTICAL TEST**
>
> If you cannot write the message format down on one page, the boundary is in the wrong place. Redraw it until you can.

CONCEPT 4 · TWO SHAPES

One general model, or several specialised ones

Both are legitimate; the choice depends on what you need to control

SINGLE GENERAL MODEL

One system does everything

WHAT YOU GET

Simplicity — one thing to deploy, monitor and update.

WHAT YOU GIVE UP

The ability to tune one behaviour without affecting the rest.

GOOD WHEN

The task is uniform, stakes are moderate, and the product is still young.

SEVERAL SPECIALISED MODELS

Each part has its own

WHAT YOU GET

Control — the right accuracy, speed and cost for each stage.

WHAT YOU GIVE UP

Simplicity. More moving parts, more monitoring, more ways to be inconsistent.

GOOD WHEN

Stages differ sharply in stakes or timing — understanding a scene versus moving an arm.

HOW TO CHOOSE, IN PRACTICE

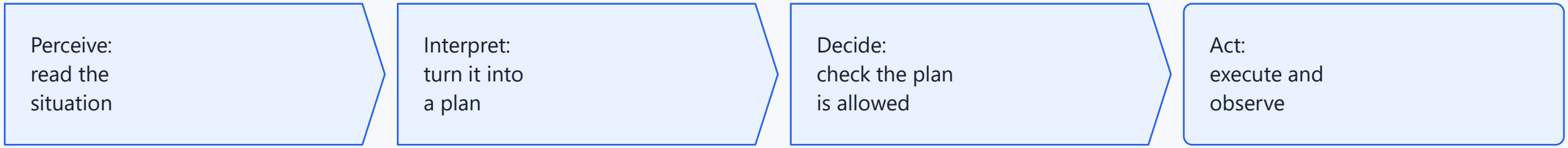
Start with the simplest thing that works, and split only when you can name the specific problem the split solves. “It feels more sophisticated” is not a reason.

CONCEPT 5 · FLOW

From what the system sees to what it does

The warehouse workflow, generalised into four stages

THE RECOGNITION-TO-ACTION CHAIN



WHERE THINGS ACTUALLY GO WRONG

Most real failures happen at the joins, not inside the boxes. A plan that is correct but arrives too late, or a confident reading of a scene the model has never seen, are both interface problems.

CONCEPT 6 · COUPLING

How tightly the parts should be joined

Both extremes cause problems, in opposite directions

TOO TIGHT

Each model assumes the internals of the next. Changing one forces changes everywhere.

LOOSELY COUPLED

Parts communicate only through an agreed message. Each can be replaced on its own.

TOO LOOSE

So many layers of indirection that nobody can trace how a decision was reached.

SIGNS YOU ARE IN THE WRONG ZONE

Symptom	What it usually means
A small model change breaks something elsewhere.	Too tight — a hidden dependency crossed the boundary.
Nobody can say which component caused an error.	Too loose — no stage records what it received and produced.
Every change needs three teams to agree.	The boundary is drawn in the wrong place.

CONCEPT 7 · FAILURE MODES

Three architecture mistakes beginners make

All three are easy to make and expensive to undo

Adding models instead of designing

Looks like A second model bolted on when the first underperforms.

Why it fails Two unclear responsibilities are harder to fix than one.

Fix Write each model's job in one sentence before adding it.

No confidence signal

Looks like A stage that always returns an answer, never an “I am not sure”.

Why it fails Downstream components cannot tell a good answer from a guess.

Fix Every model output carries a confidence, and low confidence has a defined path.

Testing only end to end

Looks like One number for whether the product worked.

Why it fails You learn that something broke, not what broke or why.

Fix Measure each stage against its own contract as well as the whole chain.

CASE STUDY

SoftBank and Yaskawa Electric

Two AI systems with different responsibilities, connected inside one warehouse workflow.

CASE STUDY · SOFTBANK AND YASKAWA ELECTRIC

A continuous workflow from recognition to action

Two models, different responsibilities, one warehouse task

THE SPLIT

Two models, two jobs

UPSTREAM

A vision-language model (VLM) running on edge computing interpreted camera images and task data, and generated subtasks.

DOWNSTREAM

A vision-language-action model (VLA) on the Yaskawa robot turned those subtasks plus its own camera input into control commands.

THE BOUNDARY

Understanding the scene and controlling the arm were deliberately kept as separate responsibilities.

THE RESULT

Recognition through to action

THE SETTING

Validation trials in a real logistics warehouse in 2026, not a fixed demonstration rig.

THE HARD PART

Object arrangements that had never appeared in training. Edge inference still produced positions and placement plans.

THE DATA

The VLM was trained on real on-site operational data labelled with grasping order and placement positions.

READING THE CASE

The architecture enabled a continuous workflow from recognition to action using integrated AI rather than fixed rules. The design decision was not to add more AI — it was to give two models different responsibilities.

APPLICATION

Bringing the theory to LumaForge Robotics

Designing the warehouse-assistant architecture

Step	What you decide	LumaForge example
List the distinct jobs	One sentence each, in plain language.	Read the shelf; plan the pick; move the arm; explain the result to the operator.
Group by requirement	Stages with the same accuracy, speed and cost needs can share a model.	Reading and explaining tolerate a second; moving the arm does not.
Draw the boundaries	Where one stage hands off to the next.	Scene understanding hands a plan to motion control, and nothing else crosses.
Write the message format	Fields, units, validity rules, confidence.	Object id, position, grasp order, confidence, plan expiry time.
Define the failure path	For each boundary, what happens when it fails.	No valid plan, or a stale plan, means stop and ask for a human.
Instrument each stage	Record what went in and what came out.	Every pick logs the scene reading, the plan and the outcome.

WHAT COMES NEXT

Session 1.2 asks where the data behind all of this comes from, and how you know it can be trusted.

SUMMARY

What to carry out of this session

1. **Give each model one job you can state in a sentence.**
   If you cannot, the boundary is in the wrong place.

2. **The interface is the architecture.**
   Message shape, validity, confidence, timing and failure behaviour — agreed in advance.

3. **Split for a named reason.**
   Different stakes or different timing justify a split; sophistication does not.

4. **Errors travel forward in a pipeline.**
   A confident misreading becomes a confident wrong action, so test each stage separately.

5. **Every stage needs a defined failure path.**
   Stopping safely is a valid output; guessing is not.

FURTHER READING

Sources for this session

Primary references behind each concept

MODULARITY	Parnas, D. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12), 1053–1058.
DESIGN RULES	Baldwin, C. & Clark, K. (2000). *Design Rules: The Power of Modularity.* Cambridge, MA: MIT Press.
PRODUCT ARCHITECTURE	Ulrich, K. (1995). The role of product architecture in the manufacturing firm. *Research Policy*, 24(3), 419–440.
ORGANISATION AND DESIGN	Conway, M. (1968). How do committees invent? *Datamation*, 14(5), 28–31.
ML SYSTEM DEBT	Sculley, D. et al. (2015). Hidden technical debt in machine learning systems. *NeurIPS 28*.
COMPOUND AI SYSTEMS	Zaharia, M. et al. (2024). The shift from models to compound AI systems. *Berkeley AI Research blog*.
PRACTICE	Huyen, C. (2022). *Designing Machine Learning Systems.* Sebastopol: O'Reilly.