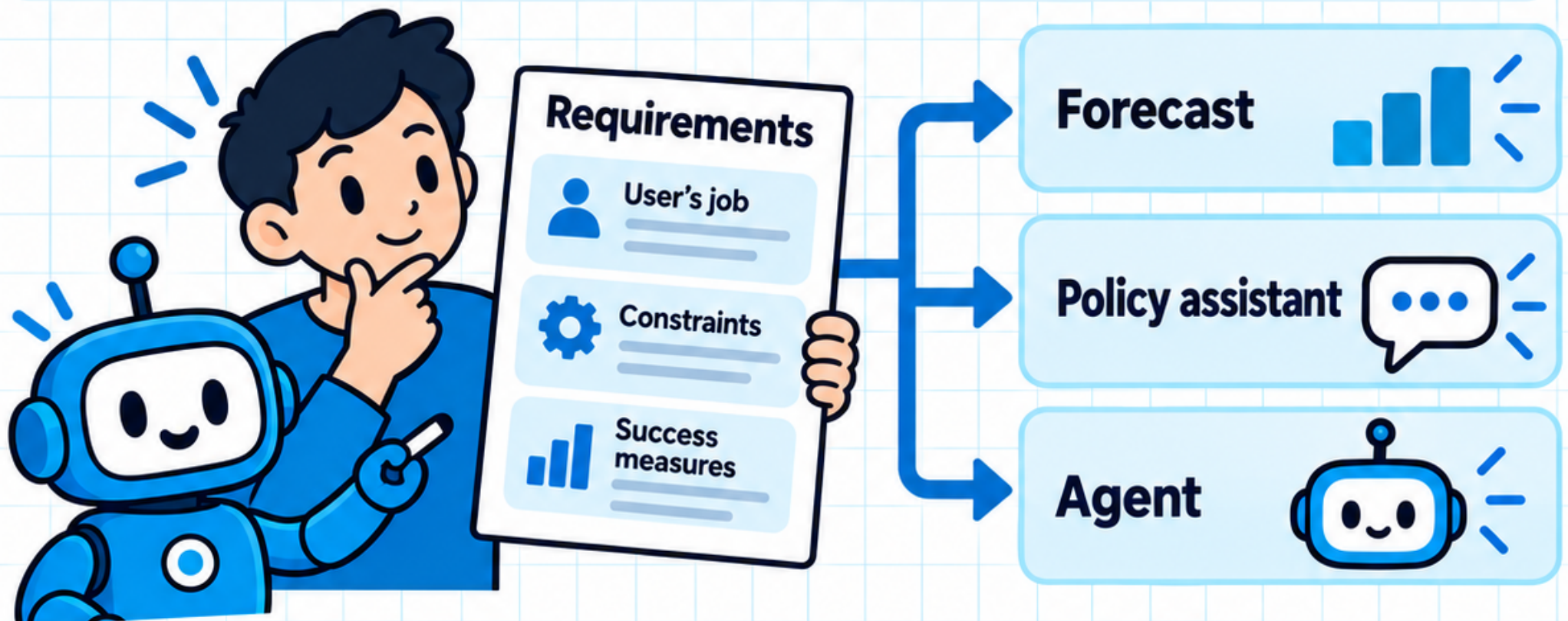


CHOOSE THE RIGHT SYSTEM SHAPE

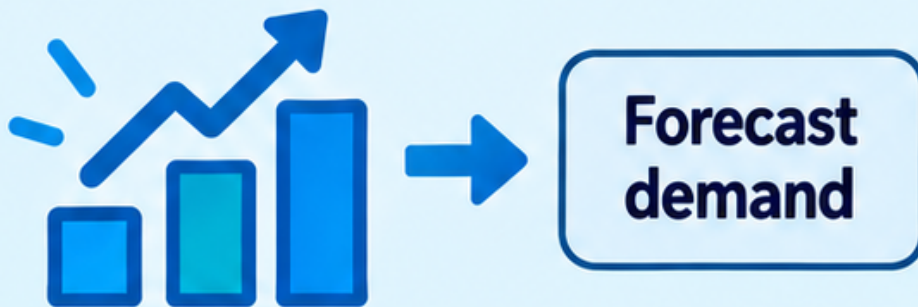
A forecast, a policy assistant, and an agent can share infrastructure. They solve different problems and need different execution paths.



Today: choose a design from the task.

ASK WHAT THE OUTPUT MUST BE

A number or score?



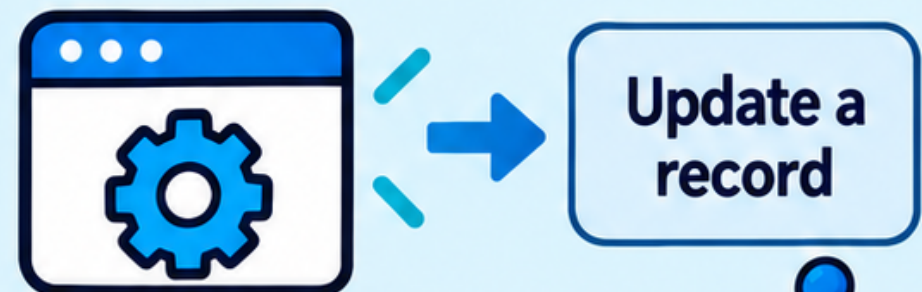
A class or extracted field?



An explanation or new content?



An action in another system?



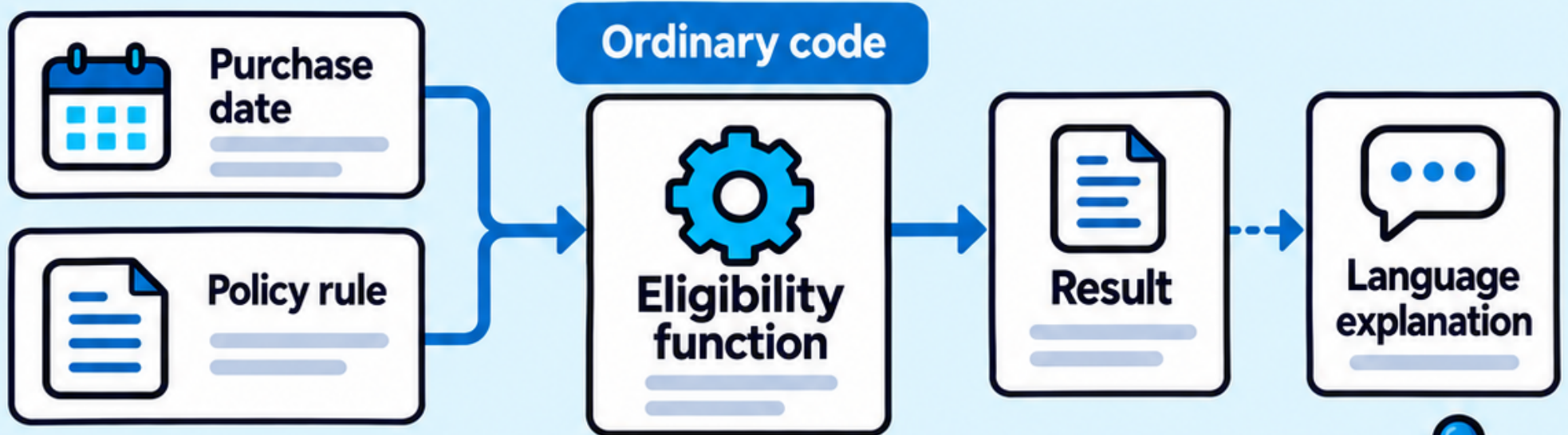
Start with the required outcome. An LLM can classify text, but that does not make every classification problem a good LLM use case.



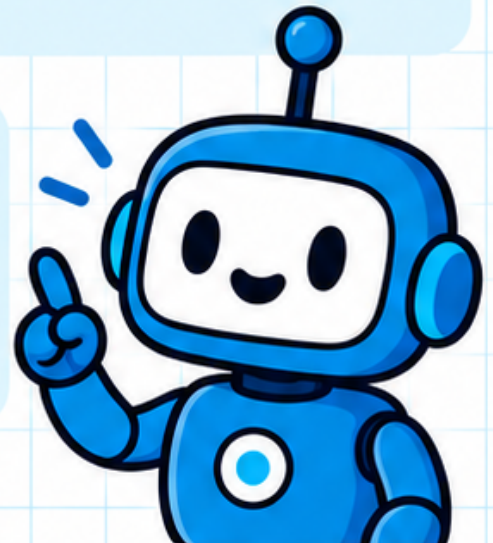
CHECK WHETHER ORDINARY CODE IS ENOUGH

Example: apply a fixed return window to a known purchase date.

If the inputs and rules are explicit, ordinary code can calculate eligibility.



A language model can explain the result. It does not need to invent or own the business rule.



USE PREDICTIVE ML FOR LEARNED PATTERNS

Example: forecast demand from historical orders, holidays, and recent activity.

Training learns a relationship from past examples. Serving applies the saved model to available inputs.

TRAINING



SERVING



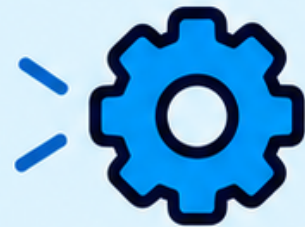
The design needs suitable data, evaluation, feature computation, model versions, and a response when inputs are missing.



BEGIN WITH A MEASURABLE BASELINE



For demand, start with a simple comparison such as the same weekday last week.



Then compare a learned model on the same future period, kept out of model fitting and tuning.



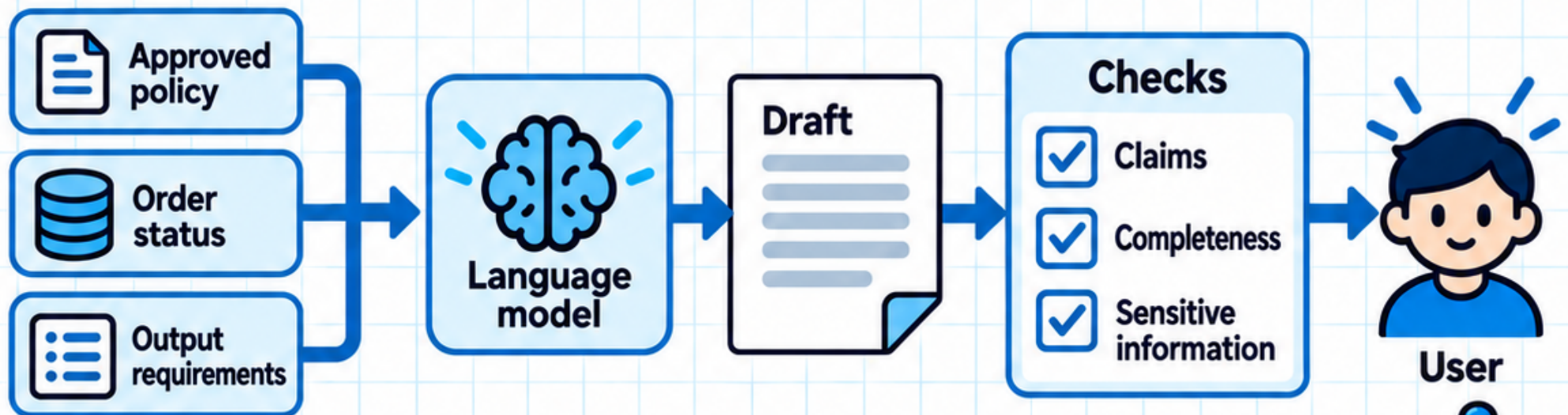
The baseline shows whether extra complexity improves the metric that matters. A complicated model does not earn its place through architecture diagrams.



USE GENERATION WHEN CONTENT IS THE PRODUCT

Example: draft a short reply from an approved policy and an order status.

The model generates language. The application supplies the facts and output requirements.



Define how to handle unsupported claims, incomplete responses, and sensitive information before sending the draft to a user.



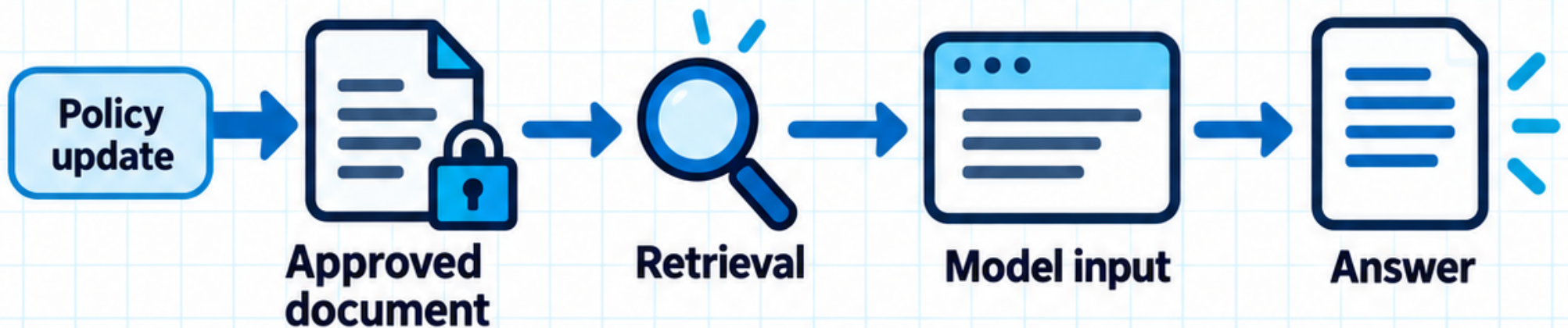
ADD RETRIEVAL WHEN THE ANSWER NEEDS SOURCES



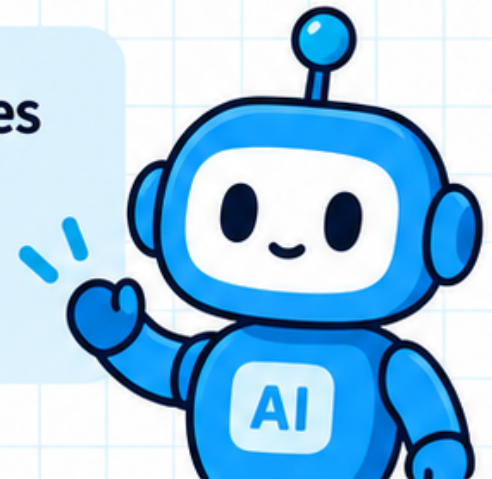
Question: “What is the current travel allowance?”



The answer depends on a controlled document that can change.



Retrieval finds permitted evidence. The model uses that evidence to answer. This is a different requirement from making the model write in a particular tone.



ADD AN API WHEN THE FACT IS LIVE



Question: “Has my order shipped?”

A policy document cannot provide the current shipment status. The application queries an authorized business API. It returns the result directly or asks a model to explain it.

Policy document



“What is the shipping policy?”

Has my order shipped?



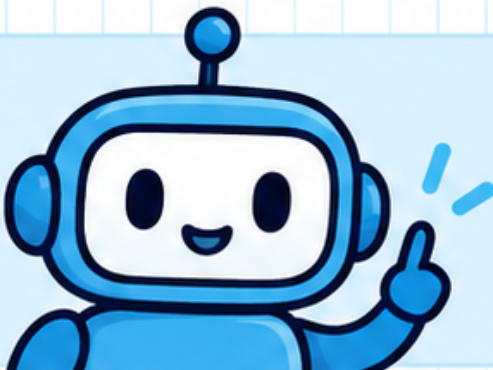
Application



Authorized
order API



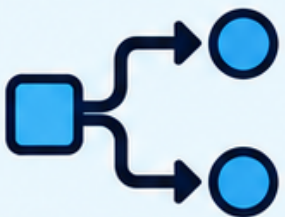
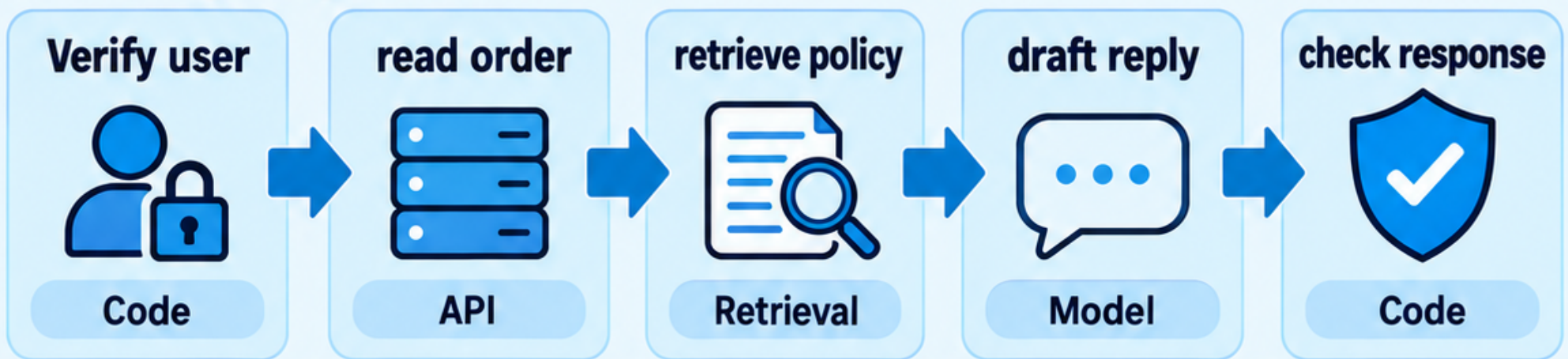
Current status



The required freshness decides the data source.

A FIXED WORKFLOW CAN COMBINE MODELS AND TOOLS

Example path:

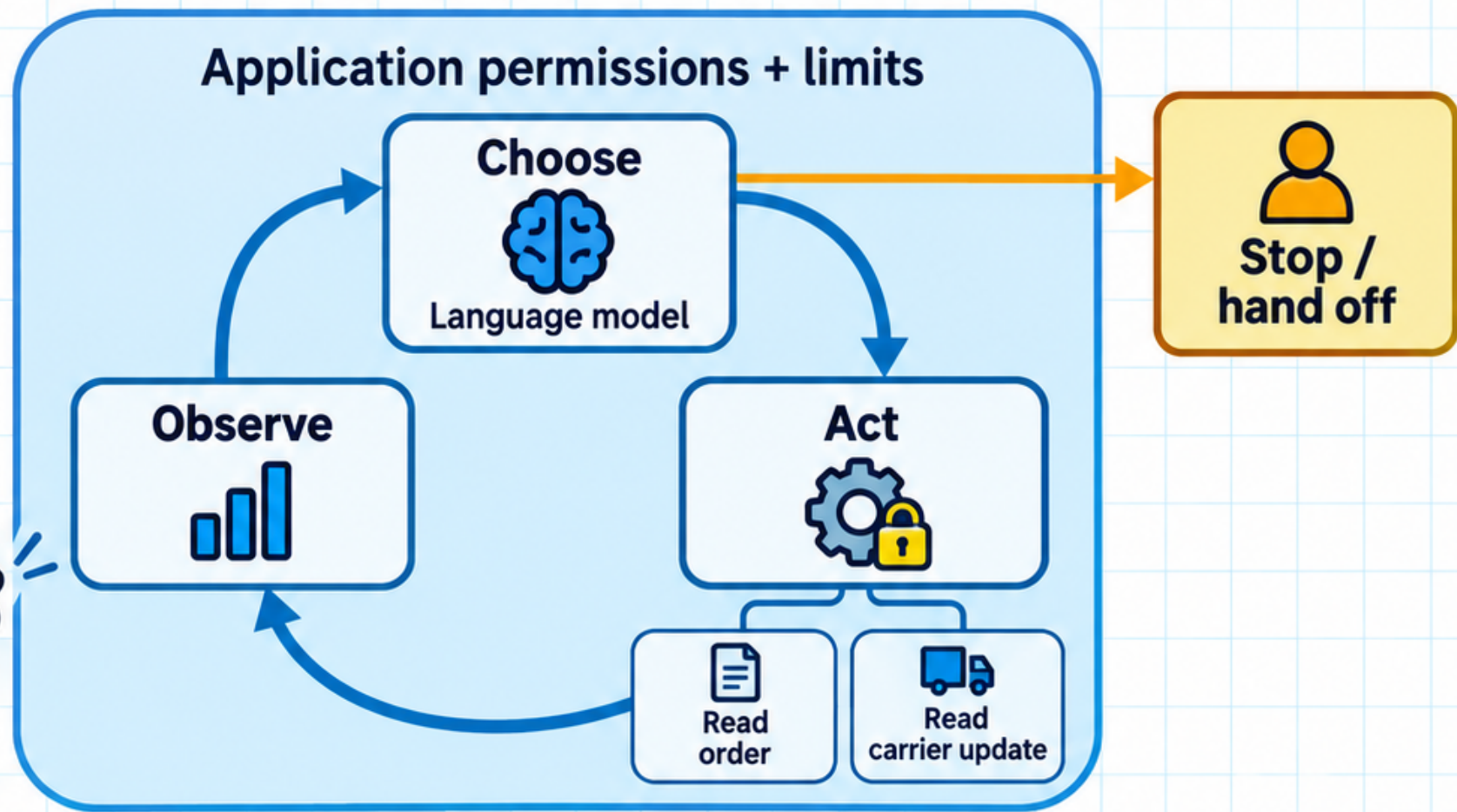
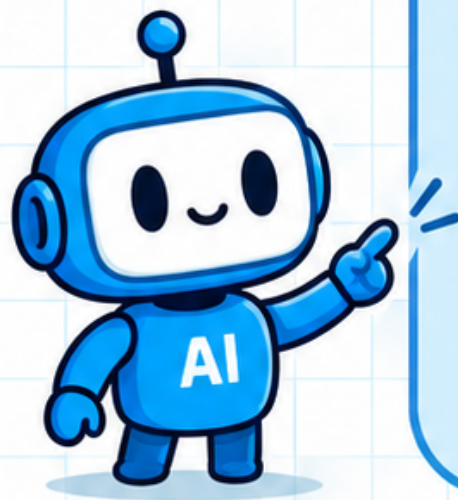


Code selects the sequence and allowed branches. Some steps use models; others use normal APIs.

Known steps make this workflow a useful baseline. Tool use alone does not require a model-controlled loop.



AN AGENT CHOOSES STEPS FROM RESULTS



Example: investigate a delayed order.

The next step can depend on whether the order exists, the carrier has an update, or the shipment needs escalation.

The model uses observations to choose the next allowed action. The application controls permissions and execution limits.

BUDGET AUTONOMY EXPLICITLY



Steps

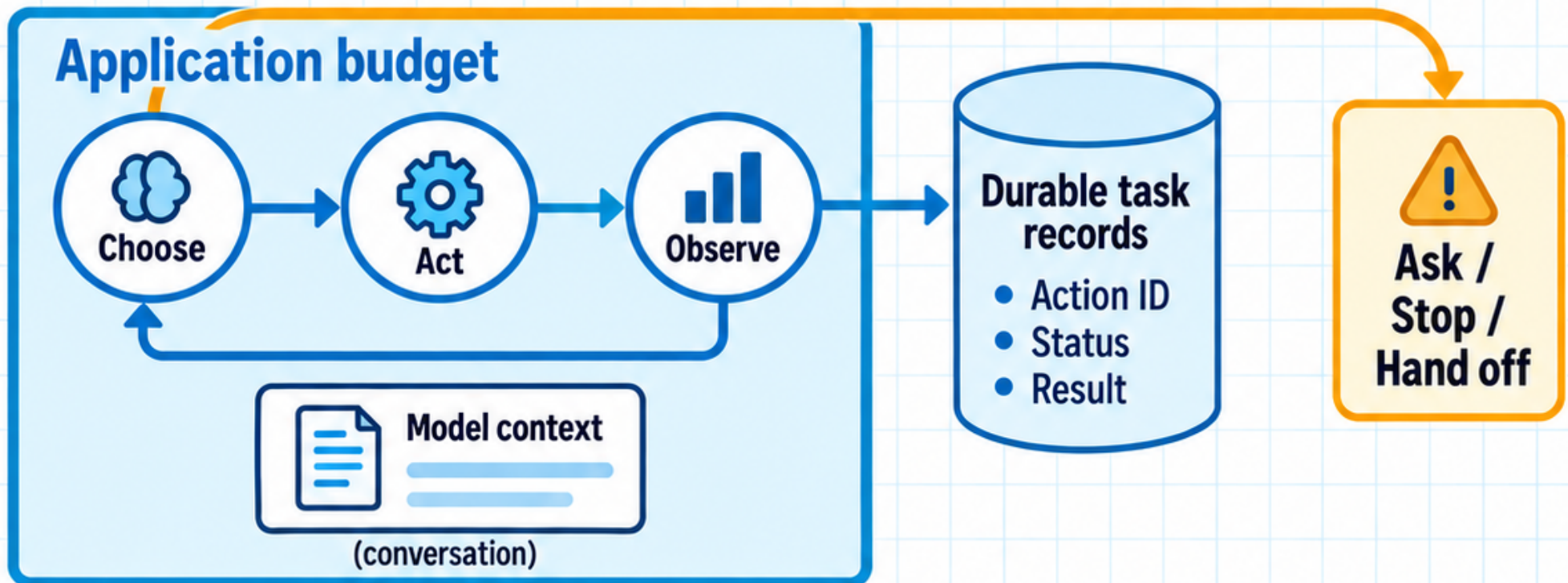


Elapsed time



Spend

- Set limits for steps, elapsed time, and spend.
- Define when the system must ask, stop, or hand off.



Store action status outside the model context. A longer conversation does not provide durable execution or prevent the same external action from happening twice.

THESE PATTERNS CAN WORK TOGETHER

A support product can use:

Support application



A classifier to route the request.



Retrieval to find policy evidence.



An API to read order status.

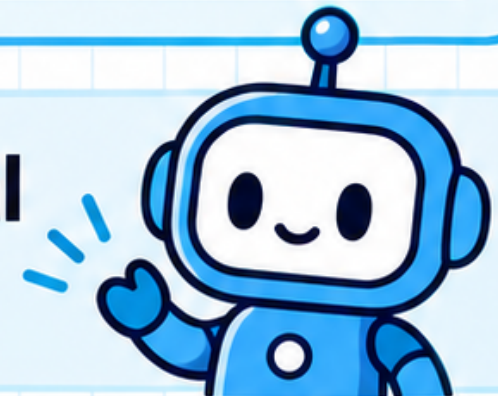


An LLM to draft the explanation.



Code to enforce refund rules.

One product can combine several model and workflow choices.

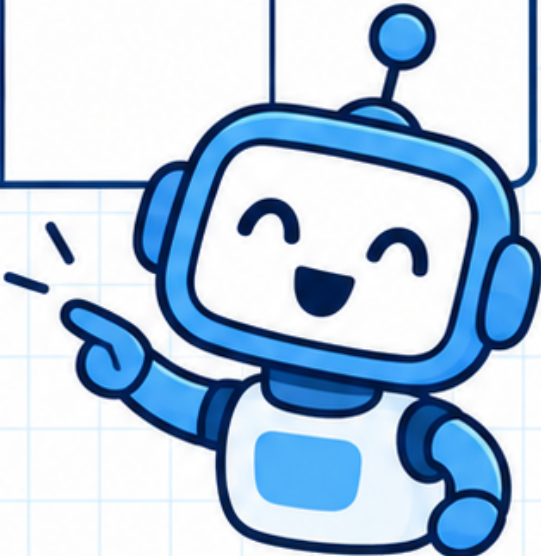


COMPARE DESIGNS ON PRACTICAL DIMENSIONS

For each candidate, ask:

	Rules	Predictive model	RAG workflow	Agent
Can it solve the required task?				
Where do its facts come from?				
Who selects the next step?				
What can it change?				
How will we measure quality, delay, cost, and failure behavior?				

A shared scorecard makes alternatives easier to compare.



Place tools after the responsibilities



scikit-learn: a predictive-model baseline.



PostgreSQL and pgvector: records and optional semantic retrieval.



FastAPI: request handling and application rules.



An LLM API: language generation or classification.



An agent framework: optional coordination for stateful model-driven steps.



A framework name is not an architecture decision.



AVOID THREE COMMON MISMATCHES

Static documents for live order status → stale answers.

✓ Use an authorized order API.



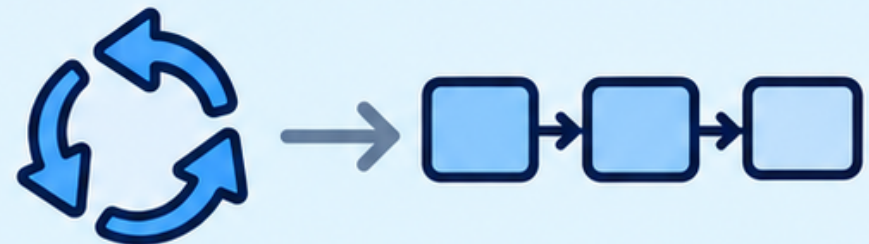
A free-form model for a fixed eligibility rule → inconsistent decisions.

✓ Use code to apply the fixed rule.



An agent loop for a known three-step process → extra operating complexity.

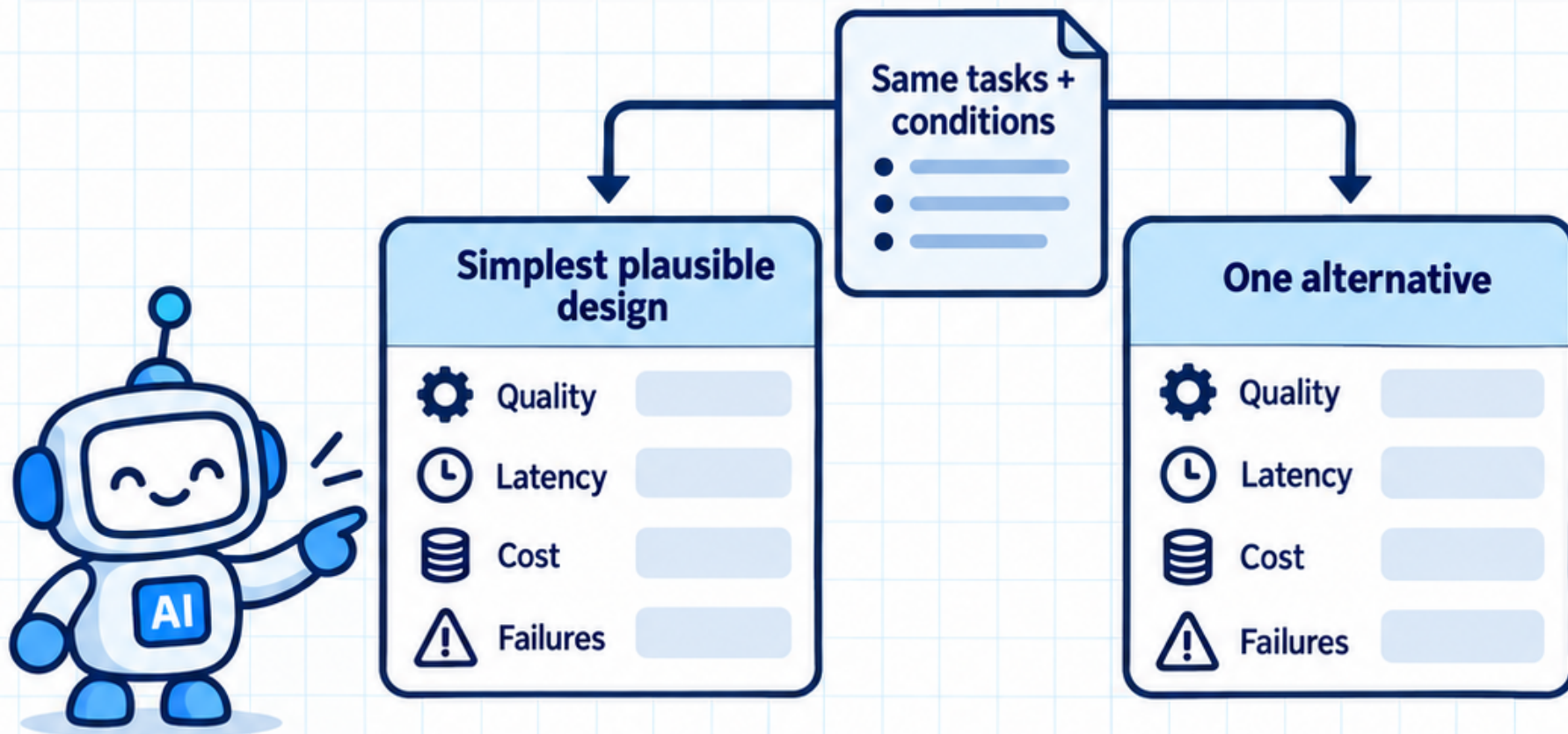
✓ Use a fixed workflow.



Fix the requirement-to-pattern match before adding another component.

RUN THE SMALLEST COMPARISON THAT HELPS

Choose representative tasks and expected outcomes.
Compare the simplest plausible design with one alternative.



Record task quality, latency, cost, and failure behavior under the same conditions.

Choose from the evidence. Revisit the decision when requirements or measured performance change.

PRACTICE WITH FOUR REQUESTS

1 Calculate a refund from this fixed rule.

Starting pattern:

2 Predict next week's demand.

Starting pattern:

3 Explain the current policy with a source.

Starting pattern:

4 Investigate why this delivery is late.

Starting pattern:

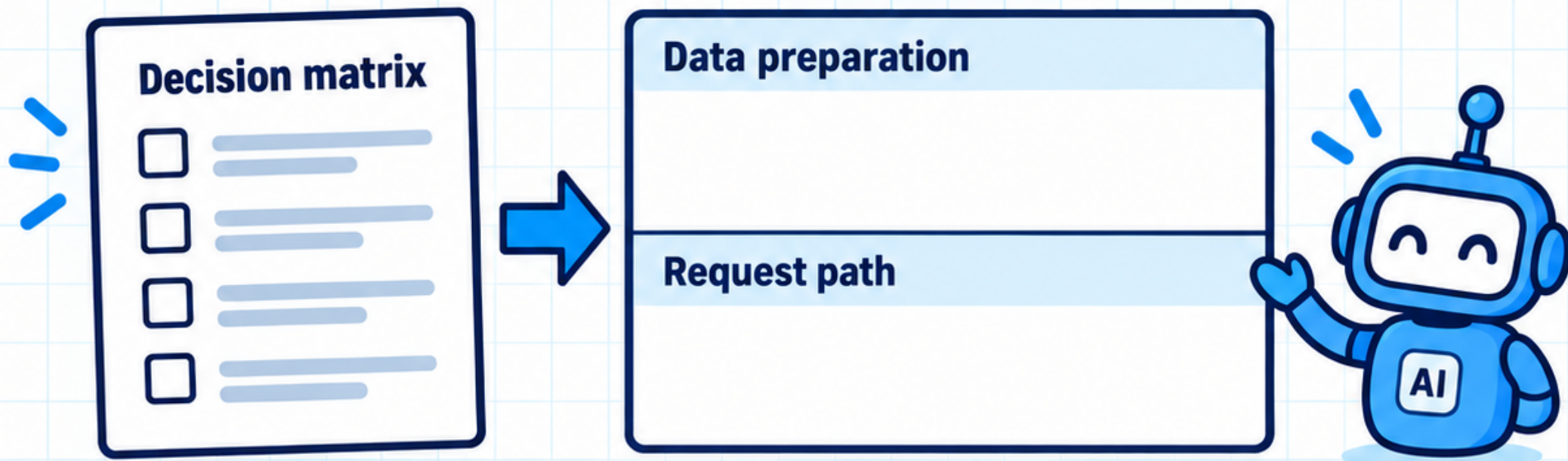
For each request, choose the starting pattern. Name the required data and one condition that changes your choice.



THE DESIGN FOLLOWS THE TASK

Rules, predictive ML, generation, retrieval, workflows, and agents are building choices.

Choose the simplest design that meets the measured requirements. Combine patterns when their responsibilities differ.



DAY 4: Data preparation and request paths

Save your decision matrix. We will turn it into a diagram next.