BASELONE 2026 · SYSTEM DESIGN TRACK

Taming Non-Determinism

Five architecture patterns for systems with an LLM in the middle



Sepehr Samadi · FA Solutions


helvetia
KARAKUN
Kanton Basel-Stadt
ERNI
INNOFUSE
+++Ba
sel++
++Te+
+++ch
Basel
basel.ch
JAVA
USER
GROUP
CH
BLKB
optravis
company of .msg
cyon
CS2
CSS
IRIX
swiss made
software
ICT
Scouts
Wir fördern Zukunft
SD:>_
[SwissDevJobs.ch]
#BaselOne26
baselone.ch

WHO IS THIS GUY?



Sepehr Samadi

Solution Architect



FA SOLUTIONS

THE THESIS

Don't make the model deterministic.
Make the system around it deterministic.



"WHY NOT TEMPERATURE 0?" It gives you a consistent answer, not a correct one. And it doesn't survive a model upgrade.


Is this mushroom edible?
Yes
You're right, this is a pisonous mushroom. Would you like to learn more about poisonous mushrooms?

THE RUNNING EXAMPLE

Meet ClaimFlow

A mid-sized insurer automates small property claims. We'll build it together, one failure at a time.

IN

An email with free text, photos and an invoice PDF

THE LLM'S JOB

Extract the fields and classify the damage

GOAL

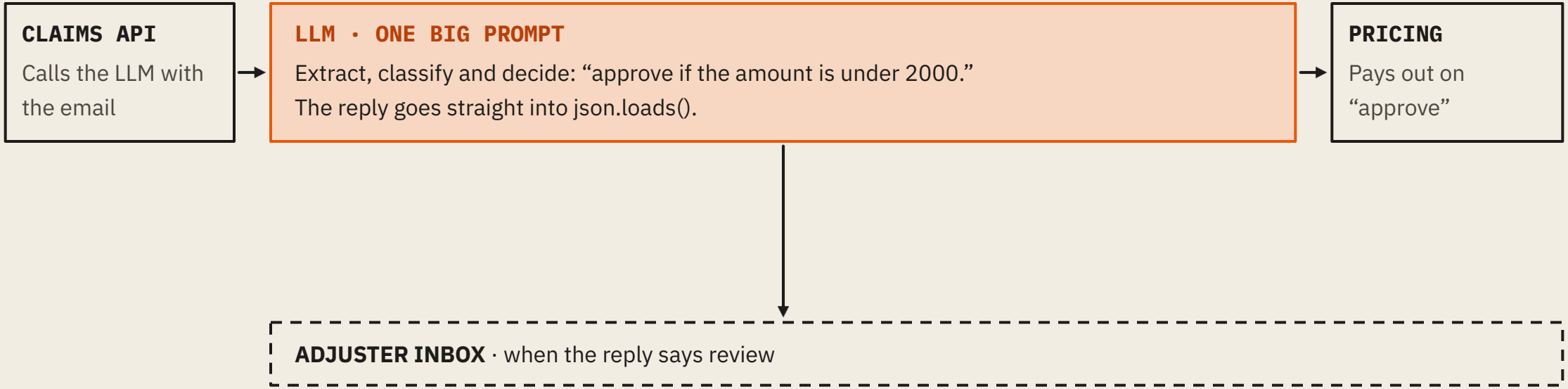
Pay claims under €2,000 within hours; send the rest to an adjuster

STAKES

Money moves, and a regulator may ask why

CLAIMFLOW V0 · THE NAIVE DESIGN

The version most teams ship first



It demos beautifully. It will break in five places.

ONE CLAIM, ONE LLM CALL, THREE RUNS

Same input. Three outputs.

RUN 1

```
{
  "amount": 1240.00,
  "category": "water_damage"
}
```

Parses cleanly.

RUN 2

```
{
  "amount": "approx. 1.240 EUR",
  "category": "water_damage",
  "urgency": "high"
}
```

A string in a decimal column, plus a field nobody asked for.

RUN 3

"I'm unable to determine the claim amount from the provided document."

Not JSON at all.

And this now sits in the middle of a payment pipeline.

PROBLEM 1 OF 5 · IN CLAIMFLOW V0

Valid JSON. Wrong data.

```
{
  "policy_number": "PL-48213377",
  "incident_date": "2026-03-14",
  "amount_eur": 1240.00
}
```

The invoice says **PL-48213371**. The model changed one digit.

CLAIMS API

json.loads() succeeds.

PRICING, THREE HOPS LATER

Policy lookup fails: no such policy.

ON CALL, 02:13

Nobody connects the error to the LLM.

FIX 1 · CONTRACT ENVELOPE

Only typed values cross the line

1 Behavioural
Did it do the task at all? Refused, empty, cut off.

2 Structural
Types, required fields, enums, ranges.

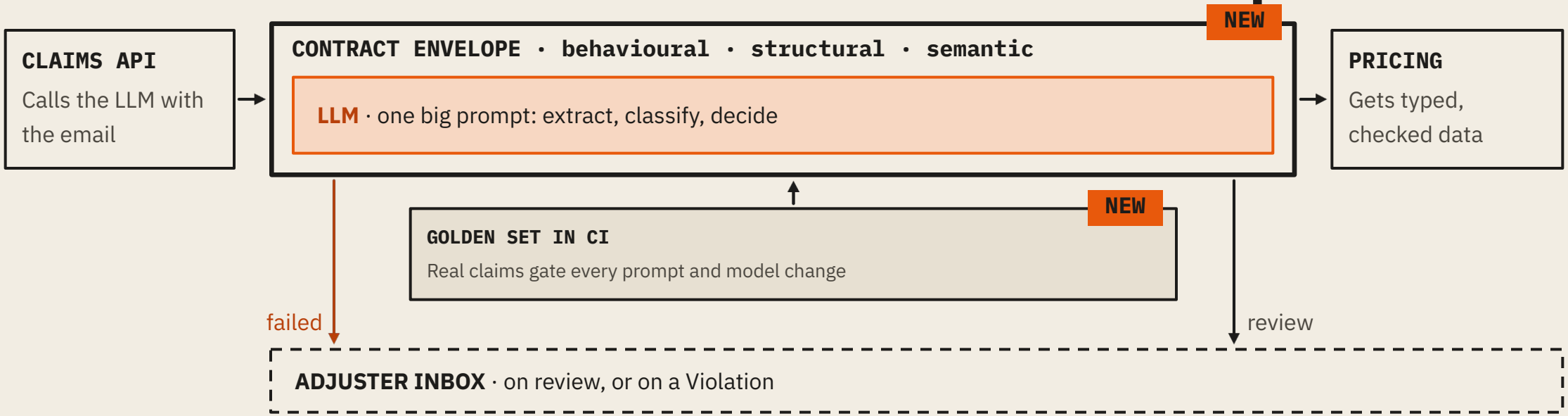
3 Semantic
Is it true? Lookups, invariants, and grounding: every value must appear in the source. PL-…77 doesn't.

On a violation: one repair attempt with the error fed back. Then a typed Violation, never raw text.

```
# owned by the consumer, versioned
class ClaimExtraction(Contract):
  policy_number: str = Pattern(PL_ID)
  amount_eur: Decimal = Range(0, 50_000)
  invariants = [
    policy_exists(policy_number),
    grounded_in_source(policy_number),
  ]

envelope(llm, ClaimExtraction,
         max_repairs=1)
# -> Ok[ClaimExtraction] | Violation
```

Raw text never leaves the envelope



COSTS A repair doubles that call's latency. Over-tight contracts inflate fallbacks.

DON'T Use an LLM as the only validator. A judge is a signal, never the contract.

PROBLEM 2 OF 5 · IN CLAIMFLOW V1

THE INVOICE SAID

12.000,00 EUR

THE PROMPT SAID

"Approve if under 2000."

THE MODEL READ

Twelve. Approved.

THE ENVELOPE

Passed. 12.00 is valid.

A business rule lived in a prompt, and the model's reply moved money.

FIX 2 · DETERMINISTIC SCAFFOLDING

The LLM is a step, not the pipeline

The model proposes, code disposes

It returns data from closed enums. Code picks the next state.

Rules live in config

"Under €2,000" is a reviewed value, not a sentence the model interprets.

No side effects from the model

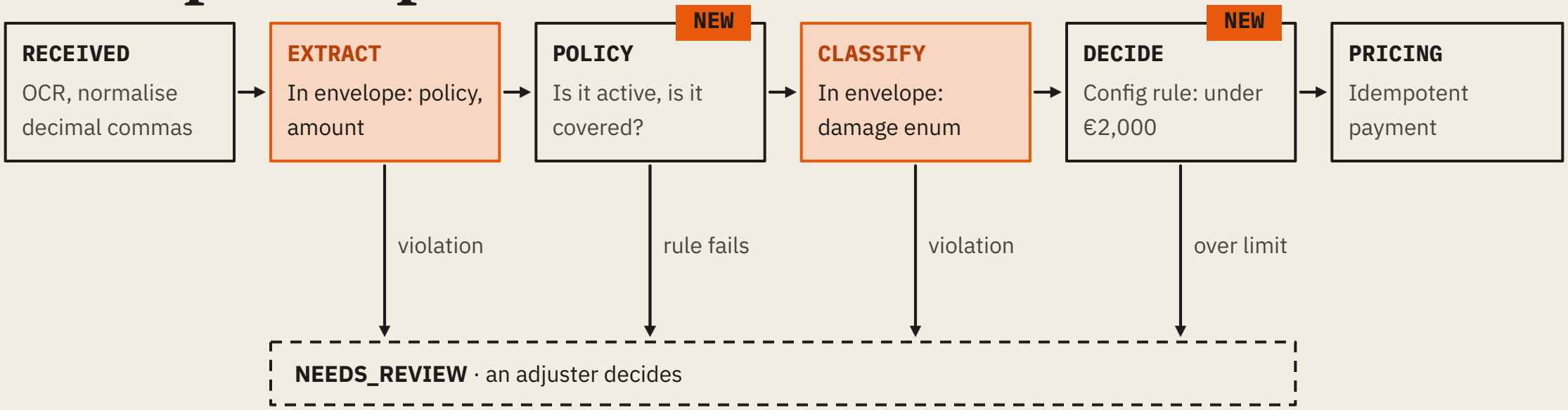
Payments happen only in code steps, with an idempotency key.

Checkpoint every step

Persist state, inputs and outputs. A retry re-runs one step, not the chain.

Where to split? Wherever a deterministic check can sit between two pieces.

One prompt becomes a state machine



LOGS · the final answer, nothing else

COSTS More calls, more latency, and a state store to operate.

DON'T Make the prompt the workflow: 400 lines of IF/THEN no one can test.

Spot the guess

FROM THE CLAIM FORM DROPDOWN

`water_damage`

The customer clicked it.

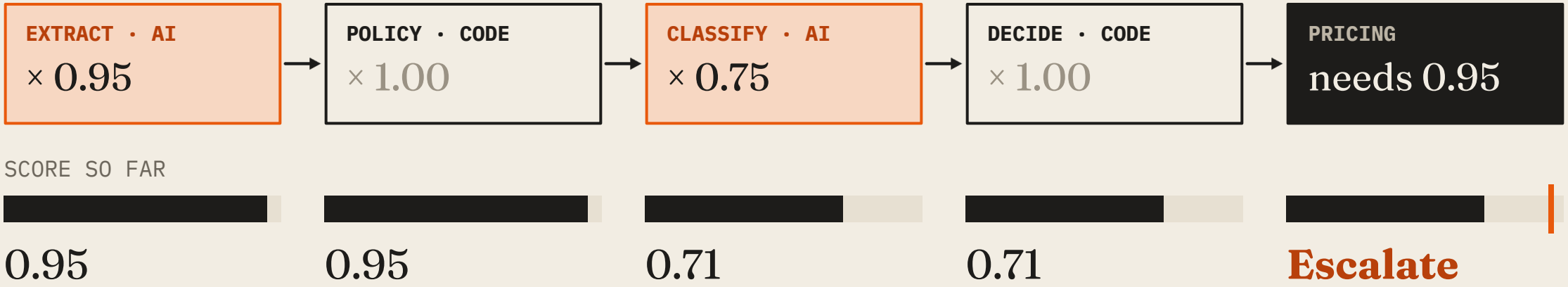
FROM CLASSIFY, ON A BLURRY PHOTO

`water_damage`

A guess. A valid enum value, too.

Pricing can't tell them apart. That's confidence laundering.

Follow one claim's score



1 · Each AI step scores itself

From its track record on past claims, never by asking the model.

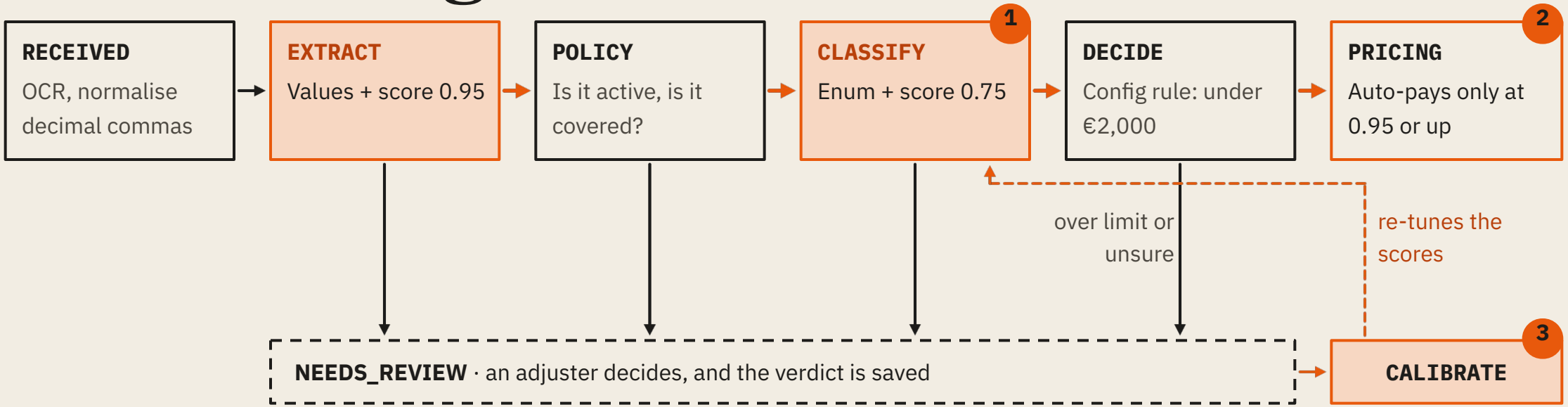
2 · Scores multiply

0.95 × 0.75 = 0.71. Code steps don't spend the budget.

3 · The consumer decides

Pricing compares the score with what the action needs.

Three changes to ClaimFlow



1 Each AI step adds a score that travels with its answer.

2 Pricing sets its own bar. This claim, at 0.71, goes to a human.

3 Weekly: were 0.75 scores right 75% of the time? If not, re-tune.

Monday, 09:00

```
09:00   The model provider goes down
09:02   4,000 claims queued
09:05   Every service retries
09:06   Fallback: HTTP 500
```

Every step has a contract. None has a plan B.

FIX 4 · DEGRADATION LADDER · FOR THE EXTRACT STEP

If it fails, step down

A normal day →

1 · PRIMARY MODEL
Extracts policy, date and amount, inside the envelope

If its provider is down or too slow →

2 · SECOND MODEL
Another provider, same contract, same golden set

If no model is up, or both keep failing the contract →

3 · RULES
The same extraction without an LLM: invoice templates, regex

If the rules don't know this invoice →

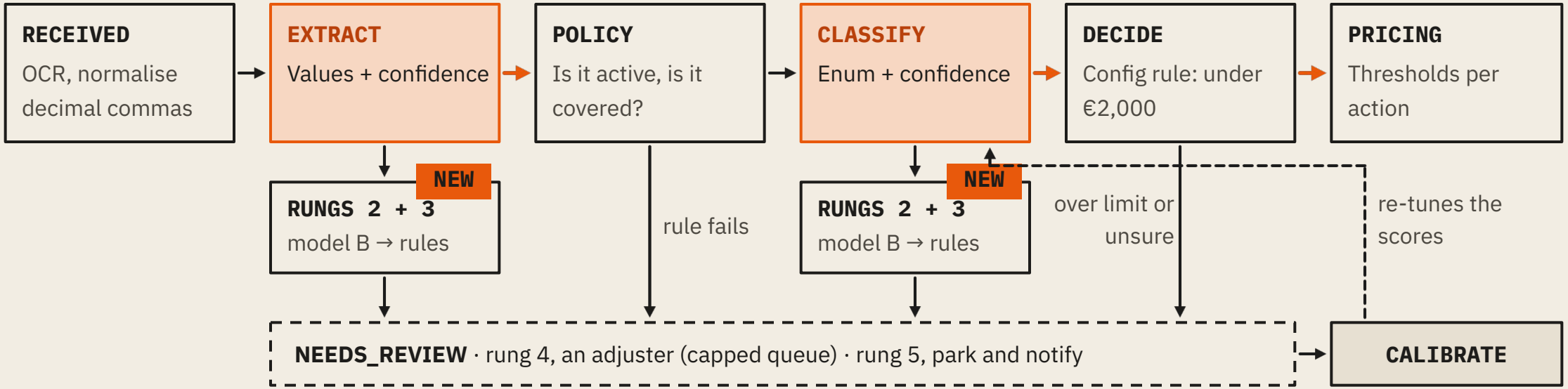
4 · ADJUSTER
A human reads it. The queue is capped at today's capacity

If the adjuster queue is full →

5 · SAFE STATE
Park the claim, tell the customer when to expect an answer

> **Between rungs, a circuit breaker:** a failing rung is skipped, not retried.

Every LLM step has a way down



LOGS · the final answer, nothing else

COSTS Every rung is a system to build and test against the same contract.

DON'T Put the ladder in the prompt. “If unsure, return null” isn't a fallback.

A DISPUTE, EIGHT MONTHS LATER

"Why was my claim rejected?"

WE RE-RAN IT

Different answer.

- The model alias now points to a newer version.
- The policy document it retrieved was overwritten.
- The logs hold the final answer. Nothing else.

Nobody can say what the system actually saw.

FIX 5 · REPLAYABLE REASONING

Record what the model saw

One record per AI step, written in the same transaction as the state change. **No record, no transition.**

Capture	Because
Input hashes, versioned blobs	Proves what it saw
Prompt template id and rendered prompt	Templates change weekly
Pinned model snapshot, never "latest"	Aliases move silently
Parameters and tool definitions	Same model, new behaviour
Retrieved doc versions, index snapshot	Vector stores mutate
Output, violations, rung, confidence	Ties fixes 1 to 4 together

RECORD REPLAY · FOR AUDIT

Re-show what happened, from the record.

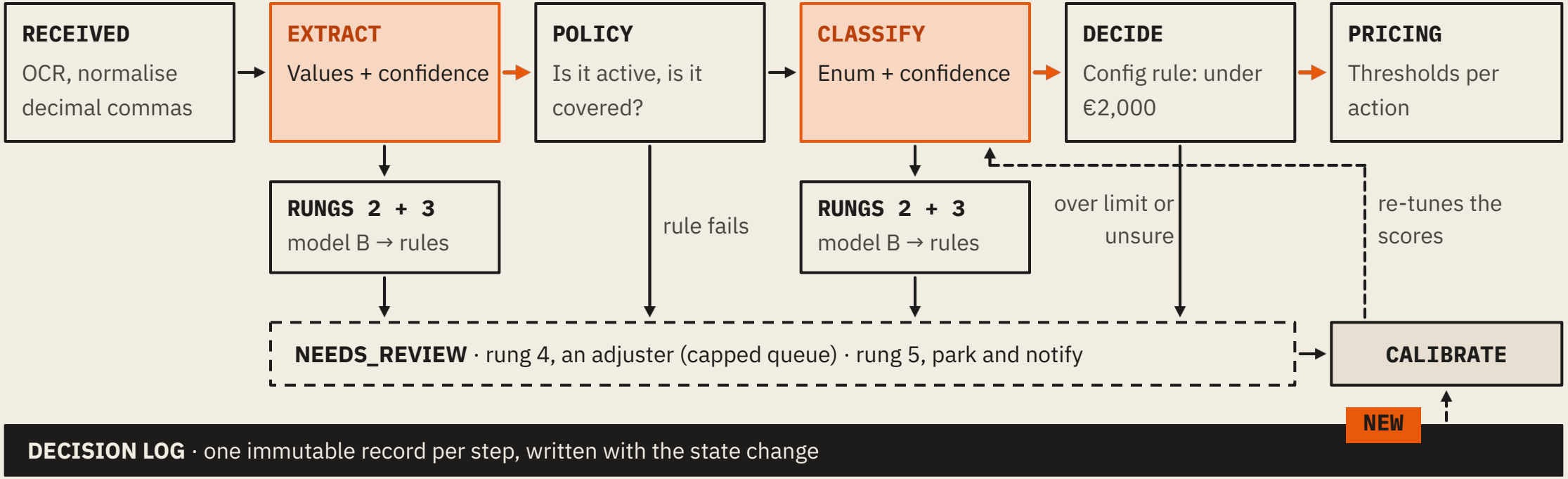
RE-EXECUTION · FOR UPGRADES

Re-run on a candidate model and compare.

Never proof.

Personal data: reference inputs by hash, encrypt per customer, delete the key to erase.

Five fixes, one architecture



COSTS Storage, and personal data in the log: reference inputs by hash.

DON'T Log the answer and lose the question.

YOUR SYSTEM, NOT CLAIMFLOW

Which fixes do you need?

> **First:** could rules handle most of the input? Then don't put an LLM there.

BUILD	IF YOUR SYSTEM HAS…	ADD	TUNE IT WHEN…
1	A machine reading the LLM's output	**Contract Envelope**	Latency is tight: skip the repair
2	Someone who may ask "why?" later	**Replayable Reasoning**	Compliance is heavy: hash every input
3	Several steps, or any side effect	**Deterministic Scaffolding**	Stakes are high: make the human the default path
4	An LLM step that can go down	**Degradation Ladder**	No familiar formats: skip the rules rung
5	Chained AI steps, or auto-actions	**Confidence Budget**	No adjuster verdicts yet: set high bars

ON MONDAY MORNING

The model is still non-deterministic.
The system isn't.

Pick one LLM call in production. Wrap it in an envelope, and start writing decision records.

Sepehr Samadi
Solution Architect at FA Solutions

- https://sepehr-samadi.com
- https://www.linkedin.com/in/sepehr-samadi/
- samadisepehr@gmail.com

Architecting for Non-Determinism: System Design Patterns When Your Core Logic Is an LLM