INTELLIGENCE WAREHOUSE · RESEARCH

The model becomes a node

Most companies already own good ML models. The demand forecast is decent, the churn score is decent, and neither one moves the business the way its accuracy says it should, because the model stands alone: a number lands in a dashboard, stripped of its error bars, blind to what other departments know, trusted equally on the day it was trained and the day the world changed under it. This paper is about what happens when a traditional ML model becomes a typed node in the Intelligence Warehouse graph. The model itself is not retrained, rebuilt, or replaced. The simulation holds the forecasts literally identical in both worlds, and the year still comes out 20% cheaper, because context, uncertainty, and drift response reach the decisions the model feeds.

Authors Akhil Singh · Nidhi Prabhu · Aniruddha Tammewar **Date** September 2026

Method Seeded simulation, shared realisation **Reproducible** seed 20260919

±0.0%	−20%	−59.1%	−92%
CHANGE TO THE MODEL ITSELF (SAME FORECASTS, SAME RMSE)	DECISION COST ACROSS THE YEAR	DECISION COST ON WEEKS WITH PROMOS, SHOCKS, OR FESTIVALS	FESTIVAL STOCKOUTS ONCE THE FIRST ONE BECAME A RULE

Read the first number carefully, because the whole paper rests on it. Both worlds receive the same forecasts from the same model with the same error. Everything that differs is downstream of the model: what the ordering decision knows alongside the forecast, how it uses the model's error history, and how fast it reacts when the model's inputs drift. The improvement is a property of the graph around the model, which means it stacks on top of whatever accuracy work the data science team does next.

01 The lonely forecast

Walk through how a demand forecast is actually consumed at most companies, and the waste has nothing to do with the model's accuracy.

The model runs on schedule and writes its numbers to a table. A dashboard reads the table. From there, the forecast's life gets rough. The supply planner copies it into a spreadsheet and adds a safety margin she has used for years, one margin for everything, because nobody ever told her the model is twice as accurate on stable SKUs as on lumpy ones. She also adjusts a few rows by hand, because she knows a promotion is coming that the model has never heard of. Her adjustments are good, and they evaporate: next week she makes them again from scratch, and no other consumer of the forecast ever benefits from them.

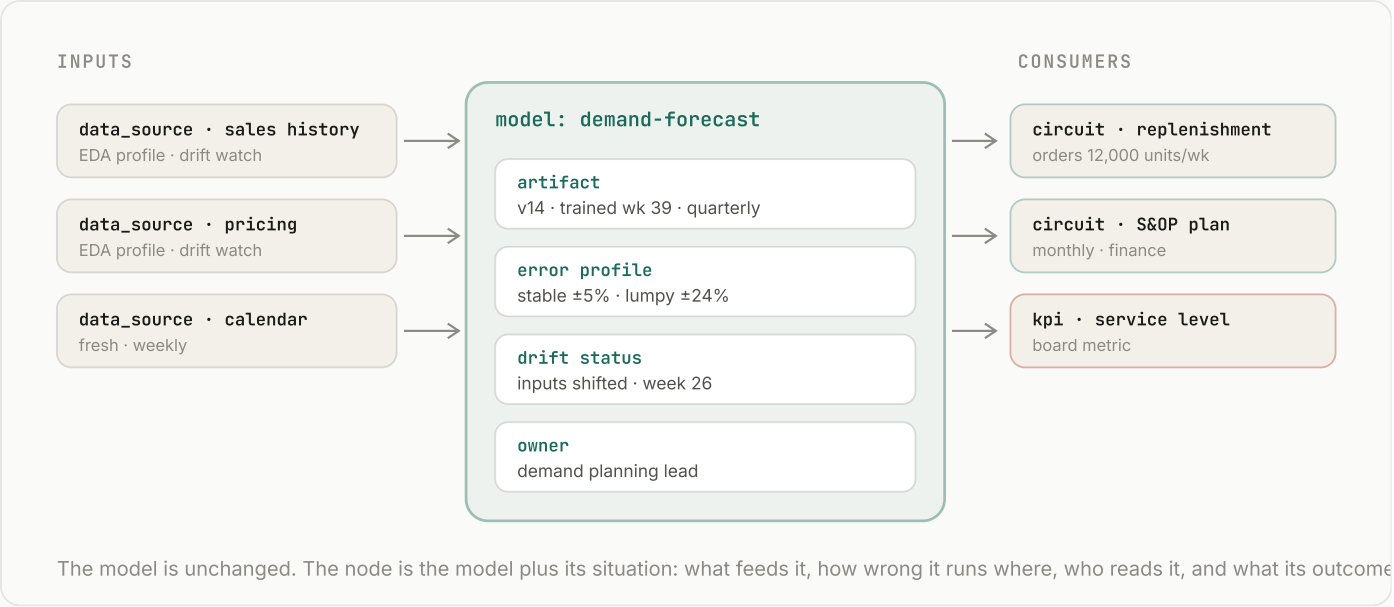
Meanwhile finance reads the same table into a different spreadsheet with different adjustments, so the company now runs on two versions of one forecast. And when the market shifts in March, the model keeps publishing numbers trained on February with the same confident face, and every consumer keeps trusting them equally until the quarterly retrain, because no signal exists that would tell them not to.

None of this is the model's fault. The model is fine. What is missing is everything around it: who consumes it, what they know that it does not, how wrong it tends to be and where, and whether the world it was trained on still exists. All of that is relational information, and the standard ML stack has no place to put it. A model registry stores the artifact. A feature store feeds it. A monitoring tool emails an engineer. Nothing connects the model to the decisions it exists to serve.

The Intelligence Warehouse treats this as the same problem it solves for every other kind of knowledge: the model becomes a typed node in the company graph, connected by typed edges to everything that concerns it. The two companion papers cover the graph itself and the decision circuits that consume it; this one covers what that placement does for the models a company already owns.

02 The model node

A model node is the model plus its situation: inputs, consumers, owner, error history, and freshness, all as edges a traversal can walk.



The model is unchanged. The node is the model plus its situation: what feeds it, how wrong it runs where, who reads it, and what its outcome

`artifact` The model itself: version, training window, features, retrain cadence. What a model registry holds today, kept, not replaced.

`inputs` Typed edges to the `data_source` nodes the model consumes. Each source carries its EDA profile and drift flags, which is what makes §06 possible: the graph knows what the model was trained on and can see when that ground moves.

`error profile` The model's measured error, kept per segment, not as one global number. Backtests and live actuals maintain it continuously. This is the model's own EDA: the system knows not just what the model predicts but how wrong it tends to be, and where.

`consumers` Edges to every circuit and flow that reads the model's output. This single piece of bookkeeping, which almost no ML stack keeps, is what turns a model problem into a routable event: when the model degrades, the graph knows exactly which decisions are exposed.

`owner` A person, with an escalation path, like any other node. Models without owners rot exactly the way documents do.

`output` Edges to the KPI nodes the model's decisions ultimately move, which is what lets retraining be prioritised by business impact rather than by global error (§06).

Notice what is not on the list: anything requiring the model to change. The node wraps the model the company already has, whether it is a gradient-boosted forecast, a churn classifier, or a price elasticity curve fit in a notebook eight quarters ago. The graph does not compete with the model. It gives the model a place to live where its output can be used with full knowledge of its character.

03 What a consumer reads

When a decision circuit's walk reaches a model node, it does not receive a number. It receives a number in context:

```
# what the replenishment circuit reads at the model node
forecast        4,180 units            # the model's output, unchanged
segment_error   ±14% (lumpy class)    # this cell's error, not the average
freshness       trained 6 weeks ago    # and the retrain schedule
drift           inputs shifted, wk 26  # flag from a data_source node
adjacent        promo: +40%, wks 30-31 # two hops away, in marketing
history         12 overrides · festival rule (week 20)
```

Every line after the first is information that exists today at most companies and reaches no decision. The error history sits in a data scientist's evaluation notebook. The promotion sits in marketing's calendar. The drift is visible in a monitoring dashboard an engineer checks. The overrides are in the planner's memory. The graph does not create any of this knowledge. It puts it on the path a decision actually walks, which turns out to be most of the value.

04 Context: what the model cannot see

A forecast model is good at patterns and structurally blind to events. The pattern is in its training data; the promotion that starts Monday is in another department's system.

This blindness is usually treated as a modelling problem, and teams spend quarters piping promotion calendars into feature stores. Sometimes that is worth it. But much of the value needs no modelling at all, because the consuming decision, not the model, is the right place for event knowledge. In the graph, the promotion is a node marketing already maintains, connected to the SKUs it affects. The replenishment circuit's walk crosses it on the way to the model node, and the order quantity gets adjusted for an event the model never saw. The model predicts the base; the graph supplies the exceptions. Division of labour, not competition.

The write-back mechanism from the circuits paper completes this. When the week-20 festival surprises everyone and the planner overrides the system's orders, that correction becomes a rule node on the replenishment circuit: festival weeks, these regions, roughly

this uplift. When week 44 arrives, the model is just as blind as it was in week 20, and it does not matter, because the rule fires. In the simulation this single mechanism cuts festival-week stockouts by **91.9%** the second time around. The planner's knowledge stopped evaporating.

05 Uncertainty: the error profile reaches the decision

Every model has an error distribution. Almost no decision downstream of a model uses it.

The standard failure is one safety margin for everything: the planner's fixed buffer, calibrated by feel to the average SKU. But the model is not averagely wrong everywhere. On stable, high-volume cells its error is a few percent; on lumpy, intermittent cells it can be five times that. One global margin therefore over-stocks the easy cells and under-protects the hard ones, simultaneously, forever.

Because the model node carries its error profile per segment, the consuming circuit can set each cell's buffer from that cell's actual error, which is elementary decision theory finally connected to its inputs: the newsvendor quantile, computed with the right distribution instead of a folk estimate of the average one. In the decomposition of §08 this mechanism contributes the least of the three, and it is worth being honest about why: better buffers help steadily but cannot rescue a decision from an event or a regime change it knows nothing about. Uncertainty handling is a complement to context, not a substitute.

06 Drift: a flag with a blast radius

Models fail slowly and quietly. The question is not whether the world drifts away from the training data, but who finds out, and how fast the finding-out reaches the decisions.

In the standard stack, drift detection exists and terminates in the wrong place: an alert in a monitoring tool, read by an ML engineer, who files a ticket for a retrain that lands whenever the pipeline schedule says it lands. In between, every consumer of the model keeps trusting it at full weight, because nothing they read carries the news.

The graph changes the routing, not the detection. The model node's input edges lead to `data_source` nodes whose profiles are already watched by the EDA machinery, so when a source's distribution shifts, the drift flag propagates along the edges: source to model to every consuming circuit. The consumers react before any retrain: they discount the model's output and lean on recent actuals until the model catches up, exactly the way a good

planner treats a forecast she has stopped trusting. And because the consumer edges exist, the flag arrives with a blast radius: which decisions are exposed, in which functions, feeding which KPIs. That last edge is what lets retraining be triaged by business impact. The model with the worst global error is not necessarily the one hurting the most valuable decisions.

In the simulation, a regime change hits a quarter of the cells at week 26, and the model retrains at week 39 in both worlds, same cadence, same model ops. The standalone world spends those thirteen weeks ordering off a fiction. The graph world detects the shift within two weeks and blends toward actuals, cutting the excess cost of the drift window by **61.6%**. Nothing about the model improved. The news simply reached the people spending the money.

07 The experiment

The simulation isolates the claim by construction. One demand realisation is generated: 200 SKU-region cells over 52 weeks, in four segments from stable to lumpy. One forecast series is generated from it, with segment-appropriate error. Then both worlds consume *the same forecasts*, so the model's RMSE is identical everywhere by construction, and only the decision machinery differs.

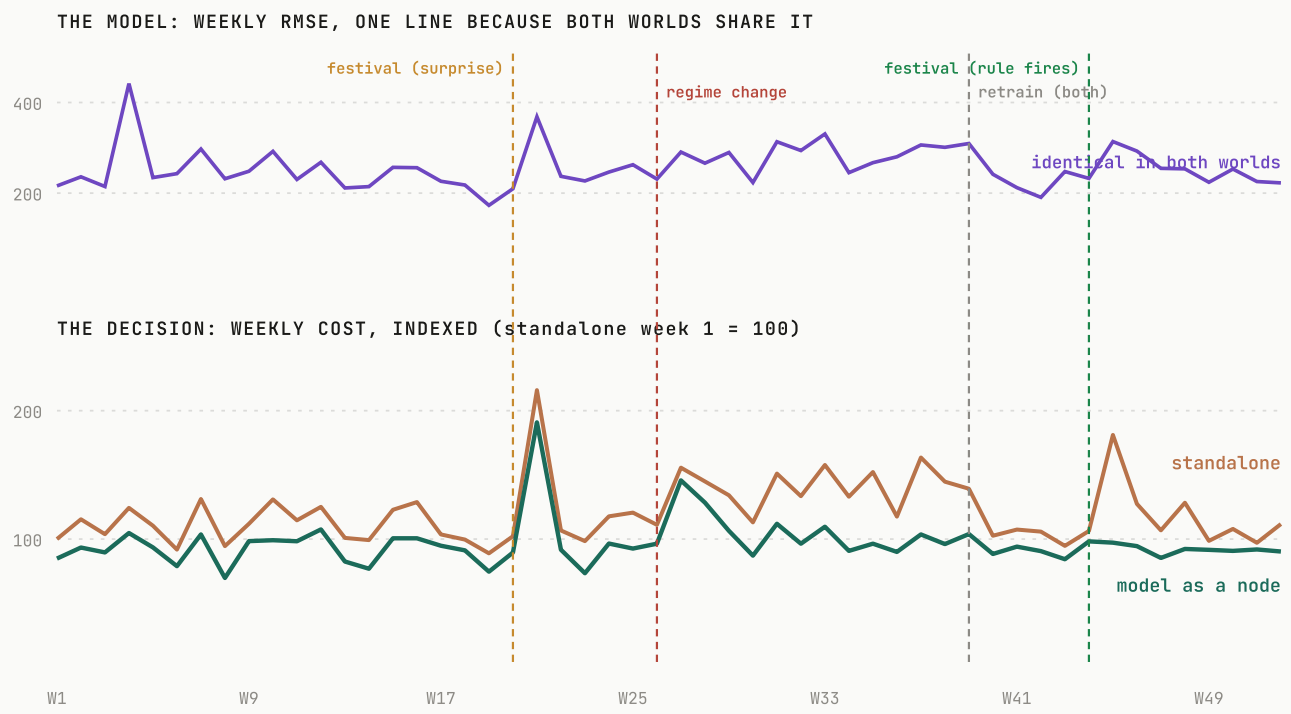
The decision is weekly replenishment with underage costing four times overage, so the right order is the forecast plus a safety quantile. The worlds:

- **Standalone.** Order from the raw forecast with one global safety factor. No event knowledge, no per-segment error, no drift signal. This is the spreadsheet-and-fixed-buffer world, competently run.
- **Model as a node.** The same forecast, plus the three mechanisms as independent switches: context (§04, promos known 90% of the time with noisy magnitudes, supply shocks 80%, festivals only after the first one writes back), per-segment uncertainty (§05), and drift response (§06, detection two weeks after the shift, blending until the same week-39 retrain).

The world also contains what the graph cannot fix: 10% of promotions arrive unannounced, known event magnitudes are estimated with 30% noise, the first festival surprises both worlds equally, and the demand noise itself is irreducible. The gap between the worlds is the part of the mess that organisation, not modelling, can remove.

08 Same model, different year

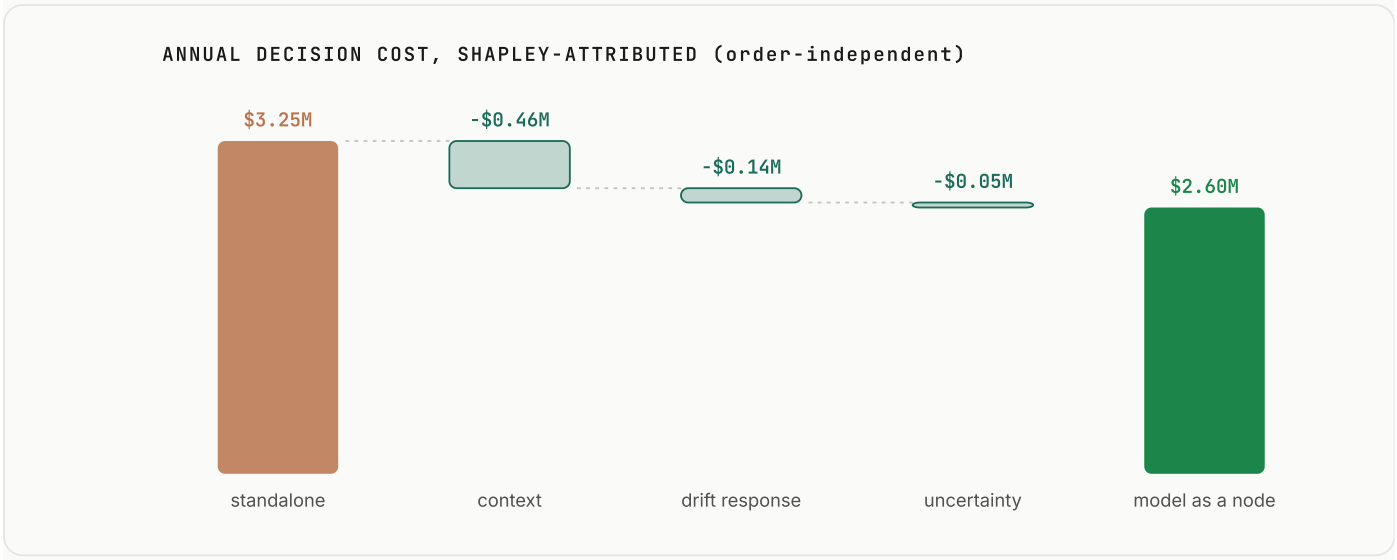
One chart carries the paper's argument. The top panel is the model's weekly error, one line, because the two worlds share it exactly. The bottom panel is what the year cost.



Same forecasts, same error, same retrain week. Only what surrounds the model differs.

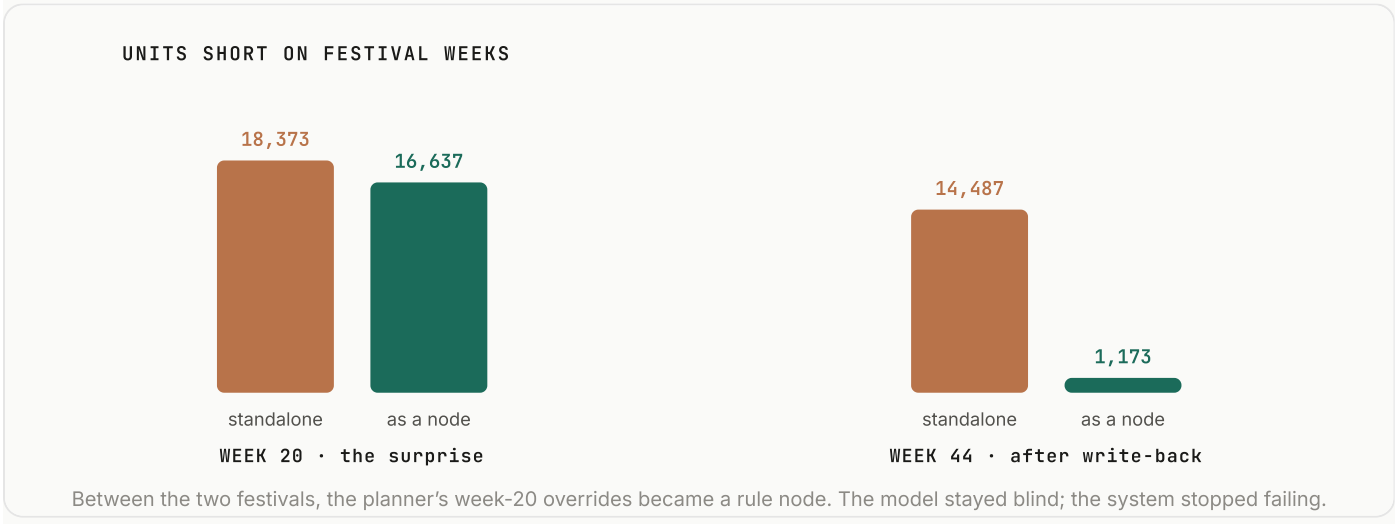
Top: weekly RMSE of the shared forecast (annual figure **261.3** units); the error spikes at the surprise festival (week 20) and through the drift window (weeks 26 to 39) hit both worlds identically. Bottom: weekly decision cost, indexed to the standalone world's first week. The gap is visible in ordinary weeks, widens at every event, and is largest between the regime change and the retrain. At week 44 the standalone world repeats its week-20 festival failure; the graph world has a rule node instead. Full year: **20%** lower cost on identical forecasts.

Where the saving comes from



Annual decision cost, Shapley-attributed across the three mechanisms (all eight switch combinations run on the same realisation, so the split is exact). **Context** carries **71%** of the saving, **drift response 21.5%**, **per-segment uncertainty 7.5%**.

The festival, twice



Units short on festival weeks. Week 20 surprises both worlds and they fail together. Between the festivals, the planner's overrides became a rule node on the replenishment circuit. Week 44: the standalone world fails again on schedule; the graph world's shortage falls by **91.9%**. The model was equally blind both times.

When does this not pay?

Two sweeps bound the claim. First, event prevalence: a business with no discrete events keeps only the uncertainty and drift mechanisms, and the saving falls to **8.6%**; at double our event rate it rises to **28.2%**. The context mechanism is worth what your calendar is

worth. Second, the cost asymmetry: at a mild 2:1 underage-to-overage ratio the saving is **14.9%**, at a harsh 8:1 it is **26.2%**. The more expensive it is to be wrong, the more the surrounding machinery matters.

EVENT PREVALENCE VS BASE CASE	COST REDUCTION	UNDERAGE : OVERAGE	COST REDUCTION
0×	-8.6%	2:1	-14.9%
0.5×	-15.8%	4:1	-20%
1×	-19%	8:1	-26.2%
2×	-28.2%		

09 One number, many functions

The quieter benefit does not show up in the simulation, because the simulation has one consumer. Real forecasts have many: supply plans against it, finance budgets from it, sales sets targets off it, and today each function reads the raw table and applies its own private adjustments, so the company runs on three divergent copies of one number and reconciles them in meetings.

As a node, the forecast has one location and every consumer's adjustment has a type. The planner's festival correction is a rule node, visible to finance's walk too. The drift discount applies to everyone at once, because it lives on the model node rather than in one team's spreadsheet. When the S&OP meeting argues, it argues about a shared object with provenance, not about whose extract is right. The one-number problem that forecast processes chase for years falls out of the architecture, because divergence has nowhere to live.

The same placement is what lets the LLM layer of the companion papers use models properly. A traversal that reaches a model node hands the language model a forecast with its error, freshness, and exceptions attached, so the narrative it writes for the Monday review ("demand up 12%, but the model is flagged for drift in the South and the festival rule fires next week") is grounded in the model's actual situation. Traditional models do the numeric prediction, which they are better and vastly cheaper at than any language model; the graph makes their numbers consumable; the LLM reasons and explains at the edges. Each layer does the one thing it is good at.

10 What this is not

The ML tooling market has adjacent products, and each solves a real problem that is not this one.

	MODEL REGISTRY	FEATURE STORE	ML MONITORING	IW MODEL NODE
Stores the artifact	yes	no	no	keeps the registry's record, as edges
Feeds the model	no	yes	no	consumes sources as typed inputs
Detects drift	no	partly	yes	inherits it from source EDA
Knows the consumers	no	no	no	every consuming circuit, as edges
Routes drift to decisions	no	no	alerts an engineer	consumers discount before retrain
Error profile at decision time	no	no	dashboards it	read by every walk
Captures consumer corrections	no	no	no	write-back to rule nodes
Retrain priority	manual	n/a	by global error	by exposed decision value

The pattern in the last column is the same one throughout: none of it is new information, and all of it is new *placement*. Registries, stores, and monitors serve the people who build models. The graph serves the decisions that consume them, and it happily coexists with all three: the node's artifact edge can point at the registry, its input edges at the feature store's sources, its drift flags at the monitor's output.

11 The honest ledger

What this costs, and where the analysis could be wrong.

WHAT THE GRAPH PLACEMENT COSTS

- **The edges must be true.** Consumer edges, input edges, and segment error profiles are bookkeeping someone has to establish and own. Most of it can be inferred (who queries the forecast table is loggable; error profiles come from backtests the team already runs), but inferred edges still need an owner to confirm them, and a wrong consumer edge misroutes a drift flag.
- **Event nodes are only as good as the function that maintains them.** The context mechanism assumes marketing's promo calendar is in the graph and roughly current. Where it is not, the mechanism quietly contributes nothing, which the §08 sweep prices.
- **Drift response needs restraint.** Blending toward recent actuals is the right move in a genuine regime change and the wrong move in a two-week blip. The simulation's detector waits two weeks and is right about the shift by construction; a real detector will sometimes fire on noise, and the blend should be sized to the confidence of the flag.

WHERE THIS COULD BE WRONG

- **The event structure is assumed knowable.** Our promotions are known 90% of the time because marketing plans them. A business whose demand shocks are genuinely exogenous (weather, virality) keeps the uncertainty and drift benefits but loses much of the context mechanism, and the zero-events sweep is the honest floor: **8.6%**.
- **The newsvendor is a clean proxy.** Real replenishment has lead times, batch sizes, and capacity coupling across SKUs. We chose the textbook decision so the mechanisms are legible; richer decisions tend to widen the gap, because coupling gives context more to know, but we have not shown that here.
- **The standalone world is competent, not strawmanned, but it is still a model.** We gave it a correctly calibrated global safety factor. A team that already runs per-segment buffers by hand has captured part of mechanism two, and their gap will be smaller by exactly that much.

12 Reproduce it

Every number on this page comes from one seeded script. The realisation is generated once and shared, all randomness is pre-drawn, and each switch combination is a pure function of it, so the decomposition is exact rather than sampled.

RUN IT YOURSELF

```
$ python3 iw-model-nodes-sim.py
# → iw-model-nodes-data.json   (every figure on this page)
# → iw-model-nodes-weekly.csv  (weekly RMSE and both cost series)
```

sim `iw-model-nodes-sim.py`: standard-library Python, no dependencies. The demand world, the shared forecast, both decision worlds, the eight switch combinations, the exact Shapley split, and both sweeps.

data `iw-model-nodes-data.json`: the aggregates this page renders from, embedded below.

seed `20260919`: change it and the draws move; the shape of every result stays.

The parameters worth replacing with your own: the underage-to-overage ratio for your actual stockout economics, your event prevalence and how far ahead your functions genuinely know about events, and your retrain cadence, which sets how long the drift window is that the graph gets to shorten.

INTELLIGENCE WAREHOUSE · RESEARCH · SEPTEMBER 2026

Authors. Akhil Singh, Nidhi Prabhu, and Aniruddha Tammewar.

The claim, in one line. Take the ML models a company already owns and make each one a typed node in the company graph, with its inputs, consumers, error profile, and freshness as edges: the identical model then produces **20%** lower decision cost across a simulated year, because decisions finally receive the forecast together with the context, the uncertainty, and the drift signals that always existed and never reached them.

Figures rendered live from `iw-model-nodes-data.json`. Simulation seed 20260919 · 200 cells × 52 weeks · identical forecasts in both worlds.