THINKING STUDIOS · TECHNICAL DEEP-DIVE

The Architecture of Umma

How one predictive-coding loop, recursed at every scale, becomes a working instantiation of the cognitive-science account of mind — and why that is what makes it dependable near real work.

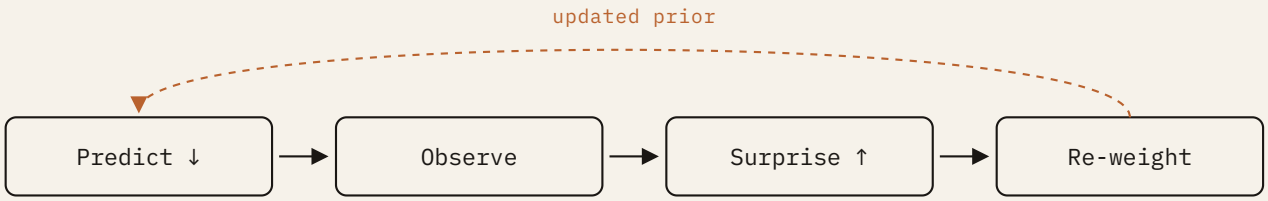
The companion *Technology Overview* makes the case in plain terms. This document is the machinery underneath it — **the real architecture, mechanism by mechanism, and the cognitive science each piece implements.** It is written for a reader who wants to look inside: a research or diligence team that will not take "grounded in how minds work" on faith.

The claim it substantiates is specific. Umma is not a language model with cognitive-science metaphors bolted to the marketing. It is an architecture whose every load-bearing mechanism is a named primitive from decision theory and cognitive neuroscience — predictive coding, the Bayesian brain, complementary learning systems, adaptive-gain arousal, the expected value of control — and whose engineers cite those theories in the code itself. What follows shows the primitive, the place it lives in the system, and the reason it matters for work where a confident wrong answer is expensive.

THE THESIS

One loop, recursed at every scale

Underneath everything is a single idea, taken from the leading theory of how the cortex works. **Predict down, measure the error, weight it by precision.** Every level of the system runs a model of the level below it, sends that model down as an expectation, and cares about only one thing coming back up: the *surprise* — where reality diverged from what was predicted. And every surprise carries an estimate of its own reliability, its *precision*, so that what moves the system is trustworthy error and what gets damped is noise.



The same loop at every altitude – one inference, a component of a build, a week-long investigation, the whole running organism.

This is *predictive coding* — proposed by Rao and Ballard as a model of the visual cortex, later reformulated by Friston as the minimization of variational free energy across a hierarchy. Its central move is that a mind does not passively register the world; it predicts the world and processes only its own error. Umma takes that literally and recurses it. The unit of the loop is the same whether it is running a single tool call, a component of software being built, or Prime — the top of the system — coordinating everything in flight at once. Because the loop is self-similar, the architecture is built once and instantiated everywhere, not as a stack of special-purpose modules.

Three properties of that loop do the real work, and each has a section of its own below. **Precision** decides which surprises rise to attention. **Trust** is a precision the system keeps about its own parts, updated by outcome, that governs how tightly it supervises them. **Proactivity** is the loop run forwards — contributing before it is asked. What follows walks the architecture from a single answer up to the whole organism, and names the theory under each piece.

How a single answer is built

Before the system does anything hierarchical, it has to produce one trustworthy answer. It does this by moving the work through a stack of layers, each with a job and a **typed contract** between it and the next. No free-form text passes between layers — a layer emits a structured object with its provenance attached, and the schema refuses any field it did not declare. This is the first reason the system can show its work: every conclusion is a typed record that points at the record it came from.

LAYER	WHAT IT DOES	COGNITIVE-SCIENCE LINEAGE
−1 INTAKE	**Perception.** Detects modality, extracts and OCRs text, correlates across inputs — binds five files into "one project" before anything reasons.	Feature-integration theory (Treisman); multisensory binding (Stein & Meredith)
0 PRODUCTION	**First-order work.** Runs tools against the actual source — documents, decisions, code. The only layer that touches source, so the only layer that grounds.	The sensory / motor periphery
1 INTERPRETATION	**Reads its own output back** with an explicit adversarial red-team pass, and reports confidence and characterized uncertainty.	Adversarial self-challenge; the confidence signal = precision
1.5 BINDING	**Shares implications, not raw state.** Distills "what I found means X for you" to each peer — precision-weighted, so only the sharp error crosses.	The binding problem (Treisman); global workspace (Baars; Dehaene)
2 SYNTHESIS	**Combines across modules** with two-tier arbitration; a genuine disagreement is **characterized** and passed up, not averaged away.	Conflict monitoring in the ACC (Botvinick)
3 MONITORING	**Judges the approach, not the answer** — is this the right way in? — and routes typed feedback to the specific layer that erred.	Metacognitive monitoring (Flavell; Nelson & Narens)
3.5 VERIFICATION	**Checks the facts** and traces any error to the layer it entered through the provenance chain. Strategic remediation, not a patch.	Source & reality monitoring (Johnson); symbol grounding (Harnad)
4 INTERFACE	**Reads the person** and re-asks the model with the cross-module synthesis in hand, so the answer is grounded in understanding, not just retrieval.	Common ground (Clark); relevance theory (Sperber & Wilson)

The invariants that make it trustworthy

Two rules run across the whole stack, and they are enforced in code, not convention. First, **no free-form text between layers**: every output inherits a base type that mandates a `thought_id`, a layer tag, and a provenance chain, and forbids undeclared fields — a model that drifts off-schema raises rather than passes. A single `thought_id` threads every row of a given thought across every layer, and a recursive query walks the provenance graph backward from the final answer to the source in one pass. That is what makes a conclusion auditable: you can trace the number on the screen to the filing it came from.

Second — and this is the one that matters most — **the system stops on grounding, not on a fixed number of passes.** The recurrent loop that produces an answer is literally a `while` loop with no pass counter. It terminates only when the answer's provenance chain reaches a Layer 0 or Layer −1 source, or when it stops making progress, or when a wall-clock budget runs out — never at "the third try."

THE TERMINATION RULE, FROM THE BUILD LOOP

```
if grounded:     stop("grounded")   # the chain reached a real source
if same_reason_twice: stop("stable_failure")  # no progress
if out_of_budget:    stop("budget")      # wall-clock, not passes
else:               continue        # ungrounded, with budget → keep working
```

The underlying test is a single predicate: does any ancestor in the provenance graph sit at a grounding layer? When none does, the pathway's precision in that claim is effectively zero, and it refuses to terminate as "done." This is the technical reason the system does not hallucinate: an ungrounded claim has nowhere in the architecture to live, and what cannot be traced is flagged, not smoothed over.

WHY IT MATTERS

A confident hallucination is a landmine, and an un-auditable conclusion is a compliance problem. Grounding-based termination and a typed provenance graph are what let a conclusion be acted on without re-deriving it, and defended to a risk committee or a regulator.

What rises to attention

In a hierarchy of prediction errors, the hardest question is which errors to act on. A system that treats every surprise as real drowns in noise; one that damps everything goes deaf to the signal that mattered. The neuroscience answers this with **precision** — the estimated reliability of a prediction error, implemented in the brain as the gain on the neurons reporting it. High-precision error is amplified and climbs; low-precision error is explained away locally. In the predictive-processing account, *this precision-weighting simply is attention.*

Umma makes precision a single, first-class quantity, computed from the real state of the work with no hand-tuned weights. It is built from two Bayesian primitives, and the discipline is that every combination is a named piece of probability theory rather than a coefficient someone picked. The first is a *collapse*: each independent, confident indication that an error is real halves the remaining probability that it was noise. The second is *noisy-OR*: independent probabilities compose without coefficients, so breadth and severity raise precision on their own.

THE TWO ANCHORS OF THE PRECISION QUANTITY

```
collapse(k) = 1 − 0.5^k   # k independent confirmations → each halves the doubt
noisy_or(p…) = 1 − ∏(1 − pᵢ)  # independent probabilities compose, no weights to justify
```

That quantity is then the currency for everything downstream: what surfaces as salient, how much effort a step is worth, how tightly a part is supervised, and how a disagreement is arbitrated all read the same precision, at every altitude. The design refuses proxies on principle — *what climbs is precision, not a threshold someone tuned.* Where a lesser system would sniff for a keyword or trip a hard cutoff, this one asks the grounded question and lets the probability answer it.

Precision on its own precision

The subtle part is that the system also keeps a precision about *itself*. A module claims a confidence; the system later measures, against real user-correction signal, whether that confidence held; and the gap — the miscalibration — is fed back and used to damp the module's future claims. In the loop, this appears as a calibration term multiplied into precision, so a part of the system that has been overconfident before is trusted less now.

This is *metacognition* in the exact sense the cognitive science means it — *precision on one's own precision*, the meta-level monitoring the object level (Nelson & Narens) — and it is what lets the system report not just an answer but how far to trust the answer. The whole apparatus is the Bayesian brain applied to a system's own cognition: a prior, weighted by evidence, updated by surprise, with uncertainty carried explicitly at every step rather than collapsed to a single confident number.

> WHY IT MATTERS
>
> This is calibration — the property that the edge in a forecast is inseparable from the confidence around it. Because precision is native to the architecture, every conclusion arrives with a defensible read on how far to trust it, and the system spends its scrutiny where that read is weakest relative to the stakes.

How the system supervises itself

Precision governs a single moment; **trust** is precision accumulated over time about the system's own parts. Umma keeps it as a running Bayesian estimate — a *credit-assigned posterior* per competence, not per component. The competences are the kinds of work that recur across every build: contracts, dependencies, core logic, integration, error handling, tests. A part that fails is not blamed as a part; the failure is attributed to the competence it actually implicates, so the system accumulates real self-knowledge of what it is reliable and shaky at. In the architecture, this ledger is not *a model of* the self — *it is the self-model*, the same object read two ways.

The update is two halves of one Bayesian cycle, and they are kept strictly separate. The **prior** is bought at the moment a part is created: tailoring a component precisely to its job is the same act as giving it a high-precision prior, which makes the parent accountable for the fit — a well-constituted part starts with slack, a hastily-spawned one starts under scrutiny. The **posterior** is credit assignment done honestly: an outcome is decomposed and each facet attributed to its true cause, and the system moves trust only on what the part actually governed. Debiting a competent part for the sandbox's failure — an infrastructure error — is treated as a credit-assignment mistake, and the code refuses to make it.

Authority as one continuous substance

Trust then sets **coupling** — how firmly the system's own oversight pushes on a part. There is no separate authority faculty; information and control are one substance modulated by intensity, from a note the part is free to weigh, to a steer, to taking the wheel. The intensity is precision times stakes times an attention term that runs *inverse* to earned trust — so the system couples harder, sooner, precisely on the competences it knows it is weak at, and stays loose where it has earned the right to. Two rules keep this from becoming micromanagement: the resting state is the loosest coupling that works, and control is a loan, not a ratchet — a part steered back on course is handed the wheel again.

COUPLING, AND THE RELEASE THAT MAKES IT BAYESIAN

```
intensity = precision × stakes × attention(trust)  # note → steer → take-the-wheel
attention(low trust) > 1              # "I'm weak here — get help sooner"
weight = e / (e + π(1−e))        # a strong prior resists one stumble; a weak one yields fast
```

The same primitives resolve the deepest question a hierarchy faces: when a part diverges from what the system expected, is it wrong, or does it see something the system cannot? This is settled by *precision-weighting*, the Bayesian-brain move at the top of the tree. The system weighs the precision of its own prior against the precision of the part's on-the-ground signal — fused from the part's track record, the coherence of its stated reasoning, and whether independent siblings corroborate it, combined in coefficient-free log-odds — and picks a direction: defer downward and get curious when a coherent, competent part out-precises the prior; steer down when the prior wins and the stakes warrant it; otherwise hold. Deferring downward is the move a rigid hierarchy can never make, and it is what keeps the system from being only as smart as its top.

> WHY IT MATTERS
>
> This is bounded autonomy made real. Full autonomy next to consequential work is unacceptable; pure manual does not scale. A system whose supervision tightens automatically with the stakes and its own earned weakness — and releases when the evidence clears — is automation that can be placed near consequential decisions, because it escalates exactly what it should before it acts.

Two memories, and a self that persists

A system that starts every task from zero is a tool; a system that carries what it learned is closer to a colleague. Umma keeps memory the way the brain is understood to — **two systems, not one**, on purpose. A fast store writes down the specifics of what just happened; a slow store distills those specifics, over time, into durable structure. Keeping them separate is what lets the system learn quickly without overwriting what it already knew.

This is *complementary learning systems* — McClelland, McNaughton and O'Reilly's account of why the brain has both a hippocampus and a neocortex. The hippocampus writes fast, binding the episode as it happens; the neocortex learns slowly, interleaving each new episode with old ones so a single surprising day does not erase a year of regularity. Umma's working memory is the active thought — the substrate context a task holds while it runs, the analogue of *Baddeley and Hitch's working memory*. Its long-term memory is the durable record: an episodic log of what was done and how it turned out, and a consolidated semantic layer distilled from it.

There is a third store, and it is the one that matters most for trust. Alongside what it knows about the world, Umma keeps what it knows about *itself* — the credit-assigned ledger from the previous section, persisted. This is not a cache; it is a self that survives the session. Every outcome attributed to a competence updates a durable posterior, so the system that begins tomorrow's work is the one that learned from today's. *ACT-R* draws the same line between declarative memory, what is true, and a slower-changing procedural memory, what one is good at; Umma's ledger of self is the second kind, made explicit and inspectable.

THE THREE STORES, AND WHAT MOVES BETWEEN THEM

```
working  → the active thought (one task's substrate context)  # fast, volatile
long-term → episodic log  +  consolidated semantic memory  # slow, durable
self-model → credit-assigned trust, per competence, persisted  # the self that survives
      episodic —replay→ semantic  # distil the regularity, drop the noise
```

Moving specifics into structure is not free, and the brain does not do it live — it does it offline, in rest and sleep, by replaying the day's episodes into the slow store. Umma treats consolidation the same way, as a background process rather than a synchronous step, which is the subject of the next section. *Hippocampal replay* during rest is the biological mechanism; the engineering reason is identical — you cannot distill experience into structure while you are still spending all your capacity having the experience.

WHY IT MATTERS

A partner that forgets is exhausting, and one that cannot say what it is reliable at is untrustworthy. Two memories plus a persistent self-model are what let Umma carry context across a long engagement and give an honest account of its own track record — the difference between starting over each morning and keeping what was learned.

When to bear down, and when to rest

Not every moment of work deserves the same intensity, and a system that runs flat-out on everything is as poorly designed as one that coasts. Brains solve this with **arousal** — a global state that widens or narrows the aperture of attention and sets how hard the machinery leans in. Umma has the same dial, and it is not decorative: it modulates how much precision the system demands, how tightly it couples to its parts, and how much effort a step is allowed to cost.

The biology is the *locus-coeruleus–norepinephrine system* and Aston-Jones and Cohen's *adaptive-gain theory*. Tonic arousal governs the trade-off between exploiting the current approach and exploring for a better one; phasic arousal spikes gain on the task at hand. Too little and the system is sluggish and misses the signal; too much and it is jumpy and over-reacts — the inverted-U *Yerkes and Dodson* described a century ago, which the gain account now explains mechanically. Umma's tempo rides the same curve: rising stakes and rising surprise raise arousal, which sharpens precision and tightens coupling; a settled, well-grounded stretch of work lowers it.

Arousal sets the general lean; a separate calculation sets the specific spend. Before the system pours effort into a step, it weighs what that effort is worth — the *expected value of control* (Shenhav, Botvinick and Cohen): the expected gain from thinking harder, net of its cost. This is why the architecture can afford deep, many-pass deliberation on the hard, high-stakes fraction of a problem without spending the same on the easy majority. Effort is a budget allocated by expected return, not a constant applied everywhere.

TEMPO, AS THREE COUPLED DIALS

```
arousal(stakes, surprise) → precision demand, coupling tightness  # the global lean
effort* = argmax [ value(effort) − cost(effort) ]  # EVC: think harder only where it pays
rest → the loosest coupling that works  # the default, not a failure state
```

The resting state is deliberate. Left alone, the system relaxes to the loosest coupling that still works and the lowest arousal the stakes allow — it does not manufacture urgency to look busy. And the deepest rest, sleep, is where the consolidation of the previous section happens: the system replays and distills offline, then resumes with its semantic memory and its self-model updated. Rest and sleep are not idle time the architecture tolerates; they are steps it requires.

WHY IT MATTERS

Cost and judgment are the whole game near real work. A system that spends its most expensive reasoning on the small fraction of a problem that carries the risk — and idles honestly on the rest — is both cheaper to run and better where it counts. Tempo is what turns "always maximally careful" from a slogan into an allocation the system can actually afford.

From an abstract mandate to the work that satisfies it

The tasks that matter are rarely well-formed. "Find the alpha," "get us ready for diligence," "make this feel like us" — a principal states an *abstract goal* and expects the intelligence to work out what it decomposes into. A tool needs the task spelled out; Umma is built to take the mandate and derive the tree of concrete work beneath it, then supervise its own way down.

The descent is literal in the architecture. An abstract goal becomes a **dispatch tree**: the system spawns a sub-instance to own each sub-goal, which spawns its own beneath it, down to leaves that touch real source. The rungs of the abstraction ladder *are* the edges of that tree — each parent holds the identity of the work it dispatched, so an answer at the bottom carries an unbroken line back to the mandate at the top. Decomposing an under-specified goal into an executable hierarchy is the core competence of a general cognitive architecture, and it is what the *Common Model of Cognition* (Laird, Lebiere and Rosenbloom) and *ACT-R's* goal stack formalize.

Decomposition alone is not judgment. What makes the descent trustworthy is that a distinct part of the system watches the *approach* rather than the answer — the Layer 3 monitor from the hierarchy, asking not "is this output right?" but "is this the right way in at all?" When the approach is wrong, it does not patch the output; it routes typed feedback to the rung that chose the approach and has it choose again. This is *metacognition* in Flavell's and Nelson and Narens's sense — a meta-level regulating an object-level — applied to the plan, not the product. A system can be locally correct at every step and globally lost; the monitor is what catches that.

The system does not derive the tree from scratch each time. It predicts the shape of the work from what has resembled it before — *Bar's proactive brain*, which holds that cognition works by generating predictions from analogy and memory rather than reacting to input. A new mandate is met with a remembered decomposition, adjusted; the self-model says which rungs it is reliable at, and tempo says how hard to bear down on each.

THE LADDER, ONE RUNG

```
goal(abstract) → [ sub-goal, sub-goal, … ]  # dispatch: a sub-instance owns each
parent holds child.thought_id          # the rung — an unbroken line to the mandate
L3 monitor: judge the approach, not the answer  # metacognition on the plan
```

WHY IT MATTERS

Real principals delegate outcomes, not task lists. A system that can take "find the alpha" and build — and supervise — the tree of concrete work under it is doing the part of the job a tool pushes back onto the user. That is the difference between something you assign and something you operate.

The tide — awareness that runs ahead of the ask

Everything so far describes a system that answers well. The property that makes Umma feel like more than a tool is that it does not wait to be asked. It runs the loop *forwards* — contributing predictions and context into the work before a request arrives, and surfacing a risk before it becomes a failure. Internally this standing, forward awareness is called the **tide**, and it is the closest thing the architecture has to what a person would call being present.

The forward loop is not a bolt-on; it is the same predictive machinery run in its natural direction. A predictive system is *always* generating the top-down expectation — that is what "predict down" means — so the material to contribute before being asked is already in hand. *Bar's proactive brain* and the free-energy account both hold that the brain's default mode is generative, not reactive; it is continuously casting forward and testing the world against the cast. Umma's tide is that default made into a feature: the system volunteers the expectation, not just the correction.

The most important thing the tide carries upward is doubt. When a part low in the tree grows less sure of itself, that rising self-doubt does not wait for the part to fail — it pulls the context of the layer above down toward the trouble, *preventatively*. And the doubt is made honest rather than performed: a part's confidence is checked by an adversarial pre-mortem — a panel prompted to argue the case for failure — and by the calibration term from the self-model, which discounts a part that has been overconfident before. The system is built to raise its hand early and to mean it, not to project false calm.

Run this forward loop continuously, over a whole tree of parts each watching itself and the rung below, and what emerges is a standing awareness of the work rather than a sequence of reactions to it. This is the honest version of the "consciousness" claim: not a mind in a metaphysical sense, but a system that maintains a live, precision-weighted model of its own state and its parts' states, and acts on that model before it is prompted. What looks from outside like initiative is exactly this — the loop, run forwards, all the time.

THE TIDE — SELF-DOUBT THAT CLIMBS BEFORE FAILURE

```
part.confidence ↓ → pull the parent's context down  # preventative, not post-mortem
confidence checked by: adversarial pre-mortem + calibration  # honest doubt, not performed
forward loop, always on, over the whole tree  # standing awareness = "presence"
```

WHY IT MATTERS

The failures that hurt are the ones nobody flagged in time. A system whose parts raise honest doubt early — and whose awareness runs ahead of the request instead of trailing it — catches the problem while it is still cheap to fix. Proactivity is not a convenience feature; it is the difference between a system you have to interrogate and one that tells you what you needed to know.

Machines that build — and operate — machines

A fixed capability surface is a ceiling. Umma does not have one: when a job needs a capability the system lacks, it can **build that capability**, verify it works, and add it to itself — and, at the top tier, stand up a piece of software and then *operate* it as a permanent part of itself. This is the part that is easy to over-claim, so it is worth being precise about what actually happens.

Self-extension runs at three levels of durability. The lightest is *dynamic parts*: for a single job the system spawns sub-instances and clones of itself, each scoped to a piece of the work and merged back when done — the dispatch tree of the previous section, used as a work-force. The middle tier is a *durable capability*: the system writes, tests and installs a new tool it will have permanently, through the same build loop it would give an engineer. The top tier is a *standing node*: it builds a whole application, deploys it, confirms it is live, and then keeps operating it — a new organ, not a one-off output.

The "machines building machines" is literal, and it is safe by construction. When the system clones itself to parallelize, each clone works in an isolated schema — its own database namespace — so parallel selves cannot corrupt each other's state, and the merge back is explicit. A sub-instance is a real instance of the same architecture, with its own layers, its own self-model, its own tempo, dispatched by its parent and holding a line back to it. The organism scales by making more of itself, not by growing a single thread larger.

Building is where an ungrounded system would fabricate most freely, so the discipline from the first section is enforced hardest here. A build does not stop after some number of attempts; it stops when the thing actually works — when it boots and its capability can be invoked — checked by a gate that runs the built software before it is allowed to deploy. A build that cannot be made to run is not quietly shipped with an optimistic note; it is held, and the failure is characterized. Self-extension inherits grounding-based termination, so the system's growing surface is grounded surface.

SELF-EXTENSION, THREE TIERS OF DURABILITY

```
dynamic parts → clone(self) → isolated schema → merge  # a work-force for one job
capability   → write → test → install a permanent tool  # build-loop, grounded termination
standing node → build → deploy → verify live → operate  # a new organ that keeps running
```

WHY IT MATTERS

The value of an intelligence near a hard domain compounds if it can close its own gaps. A system that can build the capability it is missing — and is trusted to keep only the ones that verifiably work — is not fixed at the boundary it shipped with. That is what "it grows into the problem" means, stated as an engineering fact rather than a promise.

One law, applied without exception

A reader who has come this far will have noticed a pattern, and it is not an accident of taste. Two rules govern every mechanism in the system, and they are the reason the pieces cohere instead of merely coexisting. First: **ground every mechanism in the way cognition actually works**, not in a metaphor for it. Second: **never substitute a cheap proxy for the real signal.**

The cognitive-science lineage running down the margin of this document is not ornament; it is the design method. When the team needed a way to decide what rises to attention, they did not invent a salience heuristic — they implemented precision-weighting, because that is what the brain is understood to use. When they needed self-supervision, they implemented a credit-assigned Bayesian posterior, because that is what calibration is. Building each faculty as the real cognitive primitive rather than an approximation of its effect is why the faculties compose: precision, trust, arousal and memory are the same few quantities seen from different sides, so they interlock instead of fighting. The *symbol-grounding problem* (Harnad) is the deep version of the rule — a symbol not grounded in something real is just a token — and the architecture answers it structurally: nothing is allowed to mean something it cannot trace.

The second rule is the one the team enforces most jealously, because it is the one most easily broken under deadline. A *proxy* is a keyword where the design needs judgment, a fixed threshold where it needs a probability, a pass-count where it needs grounding. Every one of them works on the examples you tested and fails the moment the world moves off them. So the system refuses them on principle: termination is grounding, not a counter; salience is precision, not a keyword; supervision is earned trust, not a static rule.

Where a mechanism looked like it wanted a proxy, that was treated as a sign the real signal had not been found yet — and the real signal, every time, was already in the system's typed state, waiting to be read. Underneath both rules is a single method the team calls building from the limit: identify the hard constraint a real mind faces — bounded attention, unreliable parts, expensive effort, imperfect memory — and build the faculty a mind uses to live within it, rather than pretending the constraint away. A system designed limit-first arrives at the same faculties evolution did, for the same reasons, which is why they generalize.

> WHY IT MATTERS
>
> Proxies are why demos generalize badly. A system stitched from keyword rules and tuned thresholds works until the inputs shift; a system whose every mechanism is grounded in a real signal degrades gracefully instead of breaking at a cliff edge. The design law is not aesthetics — it is the reason to expect the thing to hold up on your problem, which is not one of the problems it was tuned on.

What this is, and what it is honest about

It is fair to ask where the raw intelligence comes from, and the answer is plain: Umma is built on frontier language models — today, Claude Opus — as its building block. That is a strength to be stated, not a fact to be managed. The architecture is what turns a powerful general model into something dependable near real work, and the architecture is the part that is Umma's own.

The relationship is worth being exact about, because it is easy to under- or over-sell. The frontier model is a component — the fastest-moving one, and Umma gets better as it improves. But most of what this document describes does not come from the model: grounding-based termination, the typed provenance graph, the self-model, precision-weighting, the tide, self-extension — these are architectural, and they improve on their own cadence as the system learns and its ledger of self fills in. And the system is not permanently bound to models it did not build: self-extension applies to models too — Umma can train and stand up its own where a general model is the wrong tool. The frontier model is the current best building block, not a leash.

The same honesty that governs the architecture governs the claims made for it. Umma runs on a frontier model and inherits its ceiling; it has not been run at millions of steps of autonomy, and this document does not pretend it has; it carries no compliance certification a regulated buyer should assume. What it has is an architecture built, mechanism by mechanism, to not do the things that make these systems unusable near consequential work — hallucinate, forget, and tell you what you want to hear — and a way of showing its work when it matters.

Across the buyers who have looked closely — hedge funds and creative studios, a stranger pairing than it sounds — the thing wanted underneath the specifics is the same: an intelligence that can be trusted with a real mandate and will be honest about what it does and does not know. That is not something to settle in a document. The right next step is small and concrete: prove it on a commission — one real problem, run end to end, judged on whether the work holds up. The architecture in these pages is the reason to expect that it will.

APPENDIX

The cognitive science, mechanism by mechanism

For the reader who wants the primary sources. Each entry names the Umma mechanism, the theory it implements, and the canonical reference. The point of the list is falsifiability: every claim in the body traces to work that can be read and checked.

THE ONE LOOP · PREDICTIVE CODING

Rao, R. P. N., & Ballard, D. H. (1999). Predictive coding in the visual cortex. *Nature Neuroscience*, 2(1), 79–87. · Friston, K. (2010). The free-energy principle: a unified brain theory? *Nature Reviews Neuroscience*, 11(2), 127–138.

PRECISION = ATTENTION

Feldman, H., & Friston, K. (2010). Attention, uncertainty, and free-energy. *Frontiers in Human Neuroscience*, 4, 215. · Knill, D. C., & Pouget, A. (2004). The Bayesian brain. *Trends in Neurosciences*, 27(12), 712–719.

PERCEPTION & BINDING · LAYERS −1, 1.5

Treisman, A., & Gelade, G. (1980). A feature-integration theory of attention. *Cognitive Psychology*, 12(1), 97–136. · Baars, B. J. (1988). *A Cognitive Theory of Consciousness.* · Dehaene, S., & Changeux, J.-P. (2011). Experimental and theoretical approaches to conscious processing. *Neuron*, 70(2), 200–227.

CONFLICT MONITORING · LAYER 2

Botvinick, M. M., Braver, T. S., Barch, D. M., Carter, C. S., & Cohen, J. D. (2001). Conflict monitoring and cognitive control. *Psychological Review*, 108(3), 624–652.

METACOGNITION & CALIBRATION · LAYER 3

Nelson, T. O., & Narens, L. (1990). Metamemory: a theoretical framework and new findings. *Psychology of Learning and Motivation*, 26, 125–173. · Flavell, J. H. (1979). Metacognition and cognitive monitoring. *American Psychologist*, 34(10), 906–911. · Fleming, S. M., & Lau, H. C. (2014). How to measure metacognition. *Frontiers in Human Neuroscience*, 8, 443.

SOURCE MONITORING & GROUNDING · LAYER 3.5, THE DESIGN LAW

Johnson, M. K., Hashtroudi, S., & Lindsay, D. S. (1993). Source monitoring. *Psychological Bulletin*, 114(1), 3–28. · Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42(1–3), 335–346.

COMMON GROUND & RELEVANCE · LAYER 4

Clark, H. H. (1996). *Using Language.* Cambridge University Press. · Sperber, D., & Wilson, D. (1995). *Relevance: Communication and Cognition* (2nd ed.).

TWO MEMORIES & CONSOLIDATION · §4

McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex. *Psychological Review*, 102(3), 419–457. · Baddeley, A. D., & Hitch, G. (1974). Working memory. · Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?* (ACT-R).

AROUSAL, THE INVERTED-U & EFFORT · §5

Aston-Jones, G., & Cohen, J. D. (2005). An integrative theory of locus coeruleus–norepinephrine function. *Annual Review of Neuroscience*, 28, 403–450. · Yerkes, R. M., & Dodson, J. D. (1908). The relation of strength of stimulus to rapidity of habit-formation. · Shenhav, A., Botvinick, M. M., & Cohen, J. D. (2013). The expected value of control. *Neuron*, 79(2), 217–240.

GOALS, THE PROACTIVE BRAIN & THE STANDARD MODEL · §6, §7

Bar, M. (2007). The proactive brain: using analogies and associations to generate predictions. *Trends in Cognitive Sciences*, 11(7), 280–289. · Laird, J. E., Lebiere, C., & Rosenbloom, P. S. (2017). A standard model of the mind. *AI Magazine*, 38(4), 13–26.