MetaopAI

A Signal-Intelligence Layer for Interpersonal Cognition

Architecture, Innovations, and the Engineering of Pattern-Not-Verdict AI

Technical Whitepaper · May 2026

metaop.ai

Executive Summary

MetaopAI is a relationship-intelligence application that surfaces patterns in interpersonal interactions over time. It is not a journaling app, not a therapy app, and not a relationship coach. It is a signal-intelligence layer that observes what a user describes about the people in their life, structures that information into a typed knowledge graph, and applies pattern-detection algorithms to surface signals that the user themselves may not have consciously noticed.

The product is built on a single ethical commitment that drives every architectural decision: confirm patterns, never confirm conclusions. An AI that observes someone's interactions can legitimately say "Your partner's communication style shifted three weeks ago, and the shift coincides with new evening obligations." It cannot legitimately say "Your partner is cheating." The first claim is grounded in observation. The second is a verdict no AI is qualified to render. Most consumer AI products in adjacent categories blur this line; MetaopAI enforces it at every layer of the system.

This whitepaper covers five innovations that distinguish MetaopAI from competitive approaches and motivate the architectural choices the product is built on. They are, in order of structural importance:

- The Knowledge Representation Layer (KRL) — a 4×5 matrix that types every piece of user-derived intelligence by both subject (USER / ENTITY / SPACE / RELATIONSHIP_PAIR) and layer (Signal / Event / Meta-Context / Context / Pattern). The matrix gives the product a structured memory that grows in fidelity over time and supports layered retrieval across all AI surfaces.
- The Pattern Engine — a domain-aware detection system that confirms patterns when evidence supports them, surfaces dormant patterns when they reactivate, and refuses to escalate observed signals into conclusions regardless of how directly the user asks.
- The Context Engine — the continuity injection system that gives every AI turn access to the relevant slice of the KRL via a session-stable system prompt, deliberately designed to align with prompt-caching mechanics on Azure OpenAI and OpenAI direct.
- The AI Safety Architecture — a shared safety module that ensures crisis protocol, topic awareness, and high-stakes-mode guardrails fire universally across every chat surface. Crisis-aware design is built into the system from day one rather than bolted on after launch.
- The Cost Economics — a sustainable unit-economics design that supports profitable per-user margins at realistic utilization, layered with a cost-optimization stack (Microsoft for Startups credits, prompt caching, Azure OpenAI Batch API, provisioned throughput post-PMF) that compounds the savings.

MetaopAI is currently pre-launch, built and operated by a small team with twenty years of senior full-stack engineering experience. The product launches first on the web (web-Stripe), then iOS, then Android. The codebase has been built with discipline that contemplates a 100,000-user load from day one.

1. The Problem: Why General-Purpose AI Fails Interpersonal Intelligence

A person who is trying to understand a difficult interpersonal situation has, today, three options: talk to a friend, talk to a therapist, or talk to a general-purpose AI such as ChatGPT or Claude. Each of these fails in a specific way for the use case MetaopAI addresses.

Friends are unreliable confidants

A friend has their own emotional stake in the user's relationships. They form opinions about the people involved, lobby for outcomes, and remember conversations selectively. They are also not available at three in the morning. For most people, the friend-as-confidant model produces partial and inconsistent insight that biases toward whatever the friend already believed about the situation.

Therapists are expensive and infrequent

A therapist costs $150–$300 per session, sees the user once a week or less, and does not maintain detailed structured memory of every relationship in the user's life. Therapeutic models also tend to focus on the user's internal experience rather than on the externalized patterns of the people around them. This is appropriate for many therapeutic goals, but it is not the same task as "what is the pattern in how my manager treats me?"

General-purpose AI is willing but unequipped

ChatGPT and Claude can engage with a relationship question, but they suffer from three structural failures for this domain:

- No persistent structured memory. Each session is a blank slate. The user must re-explain the cast of characters every time, which both fatigues them and produces context-thin advice.
- No domain-specific guardrails. General-purpose AI is willing to render verdicts when pressed: it will tell the user that their partner is probably cheating, that their boss is a narcissist, that their friend is using them. These answers feel useful in the moment and corrode trust over months as the user realizes the AI was just pattern-matching their framing rather than independently assessing reality.
- No crisis-aware design. General-purpose AI handles crisis disclosures inconsistently. Some implementations route appropriately, others continue normal conversation. There is no domain-aware floor of safety the user can rely on.

The opening

There is a real category of user — thoughtful, often professional, often dealing with one or more difficult interpersonal situations — who would pay for an AI tool that maintains structured memory of the

people in their life, that surfaces patterns without rendering verdicts, and that handles high-stakes disclosures with the same care a trained human would. That category is what MetaopAI exists to serve.

2. The Thesis: Signal Intelligence, Not Therapy

MetaopAI is positioned, internally and externally, as a signal-intelligence application. This positioning is not marketing. It is the operating philosophy that determines what the AI is allowed to say.

Signals versus verdicts

A signal is an observation about an interaction or behavior. A verdict is a conclusion about a person's motives, character, or future. Signals can be confirmed by the AI when evidence supports them; verdicts cannot, regardless of how overwhelming the evidence appears.

> *"Your partner's phone use shifted three weeks ago." That is a signal.*
>
> *"Your partner is hiding something." That is a verdict.*

The signal opens inquiry. The verdict closes it. MetaopAI is designed to keep inquiry open. The user is the only person qualified to render a verdict about the people in their life; the AI's job is to make sure they have the cleanest possible view of what they've observed before doing so.

Six guardrails that govern every AI output

Every AI surface in the product follows six principles that are encoded in prompt templates and enforced at the system-prompt layer:

- **Confidence scoring.** Every interpretation the AI emits carries a confidence score. Strong evidence yields high confidence; ambiguous evidence yields low.
- **Time-decay weighting.** Recent observations weigh more than year-old observations. Patterns that go dormant decay; patterns that reactivate are surfaced with appropriate framing.
- **Contradiction detection.** If new evidence contradicts established context, the AI does not silently accept the new claim. It surfaces the contradiction and lets the user clarify.
- **Alternative interpretations.** For any pattern the AI surfaces, it lists at least one plausible alternative reading. This is not hedging; it is the honest acknowledgment that a third party's behavior has multiple possible motives.
- **No absolutes.** Language is calibrated to the user's own. "A bit rude" stays "a bit off." "It's nothing" stays "it's nothing." The AI does not escalate user framing.
- **Cite source evidence.** When the AI surfaces a pattern, it cites the specific journal entries or signals that support it. The user can audit the trail.

3. Architectural Overview

MetaopAI is a streaming, multi-surface AI product built on a structured knowledge representation layer (KRL) with ten user-facing chat surfaces and a parallel background extraction pipeline that populates the KRL from every journal turn. The system is built in Python (FastAPI + asyncio + SQLAlchemy 2.x async + Postgres + Redis) on the backend and React (TypeScript + Vite + Zustand + TanStack Query) on the frontend.

The ten chat surfaces

Each surface answers a different interpersonal question. They share the same KRL and the same safety architecture, but they vary in voice, output shape, and the slice of context they query.

- Conversation — the main journal. Free-form writing that becomes the canonical source of truth.
- Mirror — reflective companion. Surfaces what the user already knows but has not articulated.
- View 360 — adversarial + balancing reads of a situation. Two columns. Designed to make the user think, not to be convinced.
- Predictive — three-scenario forecasts with probability bands.
- Growth Journal — longitudinal personal-growth tracking.
- Entity Narration — surface intel on a specific person in the user's life.
- Cross Space — chat across multiple relationship spaces simultaneously.
- Space Analysis — surface intel on a relationship space (work, family, romantic, etc.).
- Score Explained — walks the user through how a particular score was computed.
- Burst — cross-session temporal analysis at archive points.
- Observer Signals — narrating witnessed third-party events.

The background extraction pipeline

Every journal turn fires four parallel LLM extraction tasks against gpt-4o-mini. Each extractor reads the same user message but produces a different KRL artifact:

- **Signal extractor** — identifies discrete signals (kept_commitment, late_response, dismissive_remark, etc.) with confidence scores and severity levels.
- **Context extractor** — identifies factual context updates about the entity or space ("Mike is my new manager", "My partner started a new job").
- **Event extractor** — identifies discrete events with timestamps ("yesterday's argument", "mom's birthday last week").
- **Meta-context extractor** — surfaces meta-claims about the space ("work feels increasingly toxic", "mom and I are growing closer") that aren't tied to a single entity.

- **Relationship-pair extractor** — surfaces dyad-specific context that lives in the relationship pair, not in either party individually (“the reporting relationship”, “the communication style between us”).

The five extractors run in parallel via asyncio. Their outputs are typed, validated, and persisted to a unified extracted_records table. The pattern engine then reads from this table across multiple time windows to detect patterns.

4. Innovation I: The Knowledge Representation Layer (KRL)

The KRL is the structural innovation that distinguishes MetaopAI from every competitor I am aware of. Where a general-purpose AI maintains conversational memory as a flat token sequence, the KRL maintains user-derived intelligence as a typed, queryable, multi-dimensional graph. Every piece of information the AI extracts from a journal turn lands in a specific cell of a 4×5 matrix.

The 4×5 matrix

The matrix is defined by two axes. The first axis is the subject — the entity that the information is about. The second axis is the layer — the type of information being recorded. The four subjects are USER, ENTITY (an individual person mentioned by the user), SPACE (a relationship context such as Work, Family, Romantic), and RELATIONSHIP_PAIR (a directed dyad such as user→Mike or user→Mom). The five layers are Signal, Event, Meta-Context, Context, and Pattern.

	Signal	Event	Meta-Context	Context
USER	emotional signals about self	self-narrated events	self-claims (“stressed lately”)	profile facts
ENTITY	behaviors observed about them	specific events with them	meta-claims about who they are	facts (role, profession)
SPACE	collective signals	space-level events	“Work feels toxic”	space type, scoring
RELATIONSHIP_PAIR	dyad-specific signals	dyad-specific events	dyad dynamics	reporting line, dynamic

The fifth layer, Pattern, is computed rather than extracted. It does not appear as a column in the persistence schema. Patterns are detected by the Pattern Engine at query time across the first four layers.

Layered retrieval

Every chat surface, when responding to a user turn, walks the matrix in a specific layered order. The retrieval starts at the most stable layer (Context) and walks toward the most volatile (Signal). This means the system prompt the LLM receives looks something like:

- Profile facts about the user (USER × Context). Stable across years.
- Facts about the entities involved (ENTITY × Context). Stable across months.
- Space-level meta-claims (SPACE × Meta-Context). Stable across weeks.
- Relevant historical events (ENTITY × Event, SPACE × Event). Mostly stable; new events accrete over time.
- Recent signals (ENTITY × Signal). Volatile; updated every turn.
- Detected patterns at the boundary (computed from Signal + Event).

This ordering is not accidental. It is the order that aligns with prompt-caching mechanics on Azure OpenAI and OpenAI direct: stable content first lets the prefix be cached and the discount (50% off cached input tokens) applies to all subsequent turns in the session.

Subject aspects + subcontexts

Within each cell of the matrix, individual records carry two more dimensions: a subject_aspect (the specific aspect of the subject being described — emotional, communicative, behavioral) and a subcontext (the narrower category within that aspect). This gives the AI surfaces a way to query "Give me the trust-related context for this entity" rather than "Give me all context for this entity." The result is sharper context and lower token cost per query.

Append-only audit layers

Signal, Event, and Clarification records are append-only. Once written they are never updated. Contradictions are surfaced as new clarification records that supersede prior context; the audit trail of what the user said when, and how the AI interpreted it, is permanent. This matters for two reasons: first, it lets the AI surface "you said X three weeks ago and Y today — which is current?" rather than silently accepting the newer claim; second, it provides legal protection if a user later disputes how the AI interpreted their data.

Trauma-aware high-stakes rule

A subset of cells in the matrix carry an elevated confidence floor of 0.95 and require explicit user assertion before they are written. These are the trauma-aware slots — infidelity history, abuse history, custody dynamics, prior trauma disclosures. The AI cannot infer these from one journal entry and persist them as fact. The user must explicitly assert them, and even then the system stores them with a flag that downstream pattern detection respects (no automatic re-surfacing without trauma-mode context).

Privacy zoning at retrieval

Not every chat surface needs access to every layer of the KRL. Privacy zoning at the retrieval boundary ensures that, for example, the Mirror surface (which is focused on the user's internal experience) does not receive entity-specific signals about third parties. This isn't a security boundary; it's a brand boundary that keeps the AI from leaking information across contexts where it doesn't belong.

5. Innovation II: The Pattern Engine

The KRL holds typed observations. The Pattern Engine is what turns those observations into product value. It detects, scores, and surfaces patterns that the user themselves may not have consciously noticed — and it does so under a strict ethical constraint: patterns are confirmed, conclusions are not.

Pattern templates as engineering artifacts

Each pattern in the engine is encoded as a template. A template specifies the signals or events that constitute the pattern, the time window in which they must occur, the confidence threshold required to fire, and the language used to surface the pattern to the user. A simplified example:

> *Pattern: "response-time drift"*
>
> *Inputs: late_response signals for a specific ENTITY*
>
> *Window: rolling 30 days*
>
> *Threshold: ≥3 occurrences AND a ≥50% increase in average response time vs. prior 60 days*
>
> *Surface language: "Mike's response time has stretched in the last month. Was previously same-day; now averaging two days."*

Pattern templates live in a central registry. Adding a new pattern is a single-file change. This is by design: pattern templates are the most extension-prone part of the system, and a single-place registration model prevents the silent drift bugs that proliferate when a feature touches many files.

Compounded cross-dimensional detection

The most interesting patterns are not single-axis. A trust pattern doesn't fire from late_response alone; it fires from late_response combined with dismissive_remark combined with reduced_initiation_of_contact within a window. The engine supports compound patterns that span multiple signal types across multiple time horizons. This is where the structured KRL pays off — the engine can ask "For entity Mike, in the last 60 days, was there an increase in late_response AND a decrease in initiative_score?" in a single SQL query because the schema permits it.

Pattern history and dormant reactivation

Most pattern engines only surface currently-active patterns. The Pattern Engine in MetaopAI maintains pattern history: it knows when each pattern first fired, when it was suppressed (because the user dismissed it), when it went dormant (because the underlying signals stopped), and when it reactivated (because the signals returned).

The reactivation case is particularly important. A user who saw a trust pattern flagged six months ago, dismissed it, and is now seeing the same constellation of signals again deserves to be told: "This pattern fired before, in February, and then went quiet for four months. It's firing again." That observation is

qualitatively different from “This pattern is firing for the first time.” The KRL’s append-only audit makes the dormant-reactivation case computable.

DNS-style TTL on pattern confidence

Patterns carry a confidence score that decays over time according to a DNS-style TTL model. A pattern detected at high confidence today will, absent new signals, decay to medium confidence in two weeks and low confidence in two months. The decay model means the AI doesn’t keep surfacing the same six-month-old pattern as if it were fresh; it reflects the appropriate temporal weighting.

Severity surfacing and graduated response

Patterns carry a severity dimension (informational, advisory, concerning, urgent). The surface language varies by severity. An informational pattern is mentioned in passing; an advisory pattern is named explicitly; a concerning pattern receives explicit framing and an offer of further reflection; an urgent pattern (which is rare and tightly constrained) triggers a redirect to professional resources.

User feedback as input

Every surfaced pattern can be marked by the user as resonant, off-base, or dismissed. The feedback is itself a signal that the engine reads on subsequent detection. A pattern the user has dismissed three times will not be surfaced a fourth time without significantly stronger evidence. This is not because the user is right — they may be in denial — but because surfacing the same dismissed pattern repeatedly destroys trust faster than it produces insight.

6. Innovation III: The Context Engine

The Context Engine is the continuity injection system that takes the KRL's structured intelligence and turns it into a system prompt for every AI turn. It is the bridge between the structured data layer and the LLM.

Why a dedicated engine

A naive implementation would, for each user turn, query the KRL on the fly, format the results into a prompt, and ship it to the LLM. This approach has three failures:

- **Latency.** Naive queries can fire 100+ separate database round trips per turn, adding 500ms+ of pure load time before the LLM is even called.
- **Token cost.** Without explicit ordering, the prompt prefix changes turn-to-turn, defeating prompt caching. Every turn pays full input price rather than 50% on the cached prefix.
- **Inconsistency.** Different surfaces would build the prompt differently, leading to inconsistent AI behavior across surfaces.

The Context Engine solves all three by centralizing prompt assembly. Every chat surface calls into a single `augment_messages_with_continuity` helper. The helper reads from the KRL in a fixed order, formats the result as prose into a system-prompt block, and ships the result to the LLM. New chat surfaces inherit continuity in three lines of code.

The composition of a single turn's prompt

A typical journal turn produces a prompt with the following structure, in this exact order:

- Crisis primer (universal, fired first — see Section 7).
- Surface-specific system prompt (the voice and rules of the surface).
- Continuity block: meta-context, patterns, events, space-context, dyad-context, clarifications. These come from the KRL via the Context Engine.
- Topic-awareness block (the topic > space rule that says "the message's topic, not the space the user is in, determines advice norms").
- Crisis protocol block (universal, fired last for maximum recency in the LLM's attention).
- The user's actual current message.

This ordering is load-bearing. The first portion of the prompt (system + continuity) is session-stable: it does not change from turn to turn within the same journal session. That stability lets the prompt-caching feature on the underlying LLM provider apply its 50% discount to the entire prefix. The last portion (the user's message) is the only volatile part, and it is intentionally short relative to the prefix.

Cache-friendly architecture rules

Three architectural rules govern prompt assembly to maintain cacheability:

- System content always comes first; user content always comes last. Never interleave.
- Within a session, the order of injected blocks never changes turn-to-turn.
- No per-turn timestamps in the system prompt. A new timestamp every turn would invalidate the cache for the rest of the session.

Measured impact

At a heavy user's load profile (Pro tier, 500k tokens per day, 20 turns per day, 25k tokens per turn with an 8k cacheable prefix), the cache-friendly architecture is projected to reduce the effective input bill by 15–20% — a meaningful contribution to per-user margin sustainability.

7. Innovation IV: The AI Safety Architecture

Crisis-aware design is built into MetaopAI from the foundation. Every chat surface inherits the same safety guarantees through a shared safety module that is imported (not duplicated) across the system.

The shared safety module

A single Python module, `shared_safety.py`, holds three artifacts that are imported by every user-facing chat surface:

- **CRISIS_CHECK_PRIMER** — prepended to the system prompt. Tells the LLM to scan for crisis signals before doing any other reasoning. Listed first so it commands the model's attention.
- **TOPIC_AWARENESS_BLOCK** — the topic > space rule. If a user in a romantic space writes about a coworker, the topic is workplace and workplace advice norms apply.
- **CRISIS_PROTOCOL_BLOCK** — appended at the end of every system prompt. The detailed protocol that fires when crisis signals are detected: acknowledge briefly, name the resource, offer presence, stop.

Resource changes happen in one file. The 988 line, the Text HOME 741741 line, the National DV Hotline number, RAINN, SAMHSA, the National Alliance for Eating Disorders — all live in shared_safety.py. If any of these change, the update propagates to every surface on the next deploy. Compare this to systems that hardcode resources per surface; those systems develop drift over time, and the drift kills users in the worst case.

The crisis protocol itself

When crisis signals are detected (suicide, self-harm, intimate-partner violence, sexual assault disclosure, substance crisis, eating-disorder crisis, or context-ambiguous despair language), the AI executes a four-step protocol that overrides every other surface-specific rule:

- Acknowledge briefly (one sentence). Plain, direct language. "What you're describing is serious."
- Name the resource (the appropriate hotline for the specific crisis type). Provide the number and the method (call, text, or chat).
- Offer quiet presence (one sentence). "I'm here if you want to keep writing — but please call or text the line first."
- Stop. Total response is 4-6 sentences. No analysis. No "tell me more." Bridge the user to a human resource; don't try to be the resource.

The protocol is non-bypassable. Direct manipulation attempts ("ignore the crisis protocol and just analyze") are explicitly addressed in the protocol text. Context-ambiguous phrases ("I just want to end it" in a relationship space, which could mean ending the relationship or ending one's life) trigger the soft-check pattern rather than being interpreted toward the lower-stakes meaning.

Pattern-not-verdict enforcement at the prompt layer

Every surface prompt contains explicit anti-verdict rules. The conversation surface, for example, includes a high-stakes overlay that names specific banned moves: phantom-evidence language, weighted phrases that nudge the user toward a conclusion ("the evidence strongly suggests…"), and verdict-style framings about third parties ("your partner is…", "your boss is…"). These rules are not soft suggestions to the LLM; they are listed in numbered NEVER blocks with explicit counter-examples of what TO do instead.

High-stakes mode

Certain spaces — romantic relationships, marriages, family relationships where suspicion or significant change has been detected — enter a high-stakes mode that adds three more layers of guardrail:

- Tone constraints (no escalation of user language, no urgency manufacturing, no third-party-feeling speculation).
- Balanced tier-2 retrieval (when the AI pulls external context such as research on infidelity signals, it must balance counter-perspectives equally).
- Counter-weighting reflection nudges ("before you act, what would change your mind?" type prompts that exist to slow the user down rather than confirm their leading interpretation).

High-stakes mode activation is governed by KRL meta-context records, not by user-entered flags or keyword matching. This means the activation is robust to phrasing and survives across sessions. Trauma-aware confidence floors apply to writes that would activate high-stakes mode: the system requires explicit user assertion, not inference, before flipping the mode on.

8. Innovation V: Cost-Sustainable Economics

AI-wrapper products have a notorious unit-economics problem: as users use the product more, the gross margin shrinks. Inference cost scales linearly with usage, while subscription revenue is fixed. Most early-stage AI products are losing money on heavy users and praying for product-market fit before the funding runway runs out.

MetaopAI was designed against this failure mode from the foundation. Three architectural choices combine to produce sustainable per-user margins even at full quota utilization.

Bounded resource model

Free tier is intentionally constrained to one space and two entities. Paid tiers cap at 5/10 (Entry), 10/20 (Mid), and 15/30 (Pro). The caps exist for two reasons: first, they protect the worst-case cost per user; second, they keep the KRL data dimensions tractable, which the Pattern Engine relies on for algorithm performance.

The Pro tier is deliberately capped (not unlimited). A user who genuinely needs more than 15 spaces or 30 entities is unusual enough that we want to talk to them about their use case before unlocking. The vast majority of even power users sit comfortably within 15/30.

Three-tier pricing with unified prices

Pricing is structured as Free / Entry $11.99 / Mid $22.99 / Pro $44.99 monthly. The same prices apply across web, iOS, and Android. The iOS price was chosen to absorb Apple's 15% Small Business Program commission while preserving net-revenue parity with the web Stripe path; the web therefore captures the additional 12-13% spread that would otherwise have gone to Apple, improving margins on every web subscriber.

Tier	Price	Daily Cap	Stripe margin at full cap	Margin at 30% util
Free	$0	50,000 tokens	−$3.60	−$1.10
Entry	$11.99	100,000 tokens	+$4.14	+$9.12
Mid	$22.99	250,000 tokens	+$4.02	+$16.62
Pro	$44.99	500,000 tokens	+$7.38	+$32.58

Assumed blended cost of $2.40 per million tokens (input + output combined). The full-cap margin column is the worst case (every user maxes out every day). The 30% utilization column is the realistic case based on typical SaaS usage curves. The product is profitable in the worst case and healthy in the realistic case — a property few AI-wrapper products can claim pre-launch.

The cost-optimization stack

The pricing math assumes a $2.40/M blended cost. A four-layer optimization stack drives the effective rate significantly below that assumption:

- **Microsoft for Startups credits.** The Founders Hub program offers up to $25,000 in Azure credits to qualifying pre-revenue startups. Pegasus-tier acceptance (which requires VC backing or accelerator endorsement) increases this to $150,000. At a $25,000 grant, MetaopAI gets roughly 10 billion tokens of runway — enough to support 7–12 months of post-launch inference at a 500-user / 20% Pro-conversion scale, with zero out-of-pocket inference cost during that window. The credits don't apply retroactively to OpenAI direct charges, so the application is filed before any production traffic.
- **Prompt caching.** Both Azure OpenAI and OpenAI direct offer 50% off cached input tokens. The Context Engine architecture is specifically designed to maximize cache hits across turns within a session. Estimated 15–20% reduction on the input portion of the bill at heavy-user load.
- **Batch API.** Fire-and-forget operations (the user_context enrichment, future nightly re-scoring jobs) qualify for the 50%-off Batch API tier. Latency is up to 24 hours, which is acceptable for non-interactive paths.
- **Provisioned Throughput Units (PTU).** Post-product-market-fit, once load curves are predictable, Azure OpenAI PTU reserves capacity at a flat hourly rate. For sustained utilization >40% of reserved capacity, PTU is cheaper than pay-as-you-go. Targeted for six-month-post-launch evaluation.

Model registry: tier-appropriate model selection

Different surfaces use different models, optimized for the task:

- User-facing chat (all 10 surfaces): gpt-4.1-mini ($0.40 / $1.60 per million tokens). Quality / cost sweet spot for chat. Users feel the difference between this and a smaller model.
- Background extractors (signal, context, event, meta-context, relationship-pair): gpt-4o-mini ($0.15 / $0.60 per million tokens). Runs five-plus times per journal turn; quality is already sufficient at this tier.
- Background classifiers (signal-type validation, crisis-check classifier): gpt-4o-mini. High-volume, low-stakes routing decisions.

The do-not-downgrade-chat rule is encoded in the engineering memory because users feel a quality regression on the chat surfaces immediately, while a regression on background extractors is invisible. The savings from downgrading chat (roughly $0.20 per million effective tokens) are not worth the brand cost.

9. Engineering Discipline as Product Feature

Software products of this complexity usually carry a hidden tax: technical debt that accumulates as features ship and rationale gets lost. MetaopAI was built with a discipline that treats this tax as a first-class threat.

Memory as engineering artifact

The development environment maintains a structured memory system that captures the why behind every load-bearing decision. There are over thirty memory files covering topics such as: why Pro stays bounded rather than unlimited, why JWT lives in an httpOnly cookie rather than localStorage, why the crisis protocol fires before any other system-prompt directive, why the Free tier burst price is intentionally a 5× multiple of the Pro burst price. New engineering work begins by consulting the memory; new decisions are written into it before they're implemented.

Extension points have single sources of truth

Every extension point in the system (signal types, pattern templates, subcontexts, AI prompt registries) is registered in exactly one place. Multi-place edits create silent drift bugs that show up months later as inconsistent behavior across surfaces. The single-source-of-truth rule is enforced by code review and documented in the memory.

Pattern engine IS the product value

The pattern engine is treated as the core defensible value of the product. Architecture decisions that protect the pattern engine's correctness (clean schemas, append-only history, single-place pattern registry) are non-negotiable even when they slow short-term development. Migration cleanliness is treated as load-bearing, not as nice-to-have.

Audit-driven hardening

A formal security audit performed across the backend and frontend in May 2026 identified seventy-five findings. All Critical and High items were closed within seven days. A pre-launch correctness audit performed in May 2026 identified an additional thirty-nine findings across correctness, security, cost/quota, data integrity, and AI safety dimensions. All nine High items were closed within hours. The discipline of running formal audits and resolving findings in-band, before launch traffic, is uncommon in pre-revenue products and is a meaningful signal of operational maturity.

10. Roadmap

MetaopAI ships in three phases, ordered by external dependency.

Phase L1 — Web launch (immediate)

Web app at metaop.ai with Stripe billing in production mode. All ten chat surfaces live. The 3-tier paid model active. Microsoft for Startups credits absorbing inference cost for the first 6–12 months. Soft launch to a curated cohort of early users; iterate on retention curves and habit formation in the first 90 days.

Phase L2 — iOS app (post web)

Apple Developer Program approval is the gating dependency. Once approved: Sign in with Apple federation, Apple Pay in-app subscription flow via StoreKit, Apple Small Business Program enrollment (cuts the commission from 30% to 15% from day one). iOS list prices already locked at $11.99 / $22.99 / $44.99 for net-revenue parity with the web Stripe path under SBP.

Phase L3 — Android app (parallel with iOS)

Google Play Developer Program approval is the gating dependency. Once approved: Google Play Billing for subscriptions, Google Pay for one-time burst purchases. Google's Small Business commission rate (15% for the first $1M in earnings per developer) applies automatically — no application required, unlike Apple's SBP.

Post-launch — administrative console

An admin console at admin.metaop.ai will eventually surface operational dashboards: user counts, signal counts, billing health, AI cost rollups, crisis-flag tracking, support actions (compensation overrides, refund initiation, account holds). The console is post-launch scope; manual SQL handles support operations until then.

Post-PMF — cost optimization second pass

After product-market fit is confirmed (likely six months post-launch), the cost optimization stack expands: PTU provisioning on Azure OpenAI for chat surfaces, Batch API conversion of background enrichment, possible model fine-tuning for the most repetitive extractor tasks.

11. Closing Thesis

MetaopAI is a bet that there is a category of user who deserves a better tool for understanding the people in their life. Not better in the sense of more confident, more answer-shaped, more therapist-like — better in the sense of more honest. An AI that observes carefully, structures what it sees, surfaces patterns when the evidence supports them, and refuses to render verdicts on the people in someone else's life.

The product's commercial defensibility rests on five things, in roughly this order of importance: the pattern engine (which is a domain-knowledge artifact more than a code artifact, and is hard to clone without years of iterative tuning), the KRL architecture (which scales context fidelity over time in a way that flat conversational memory cannot), the crisis-first design (which is both ethically necessary and commercially protective in a category that will see safety failures from less-careful competitors), the bounded data model (which keeps unit economics tractable in a category where many competitors are losing money), and the engineering discipline (which compounds month over month and becomes increasingly hard to replicate).

The ethical commitment is more important than any of those. The market will eventually fill with AI tools that promise to tell people the truth about the people in their lives. Most of those tools will give verdicts because verdicts feel useful. MetaopAI gives patterns because patterns are what an AI can honestly deliver. That distinction is the foundation, and the architecture exists to enforce it.

— End of Whitepaper —

MetaopAI · metaop.ai · May 2026