Table of Contents

MetaopAI — Architecture

Schema, pipeline, latency profile, eval roadmap, deployment specifics

Technical Reference · May 2026 · metaop.ai

> **Companion document.** For the philosophical / positioning case, see WHITEPAPER_INVESTOR.md. This document assumes the reader understands the product premise and wants the implementation specifics.

Table of Contents

---

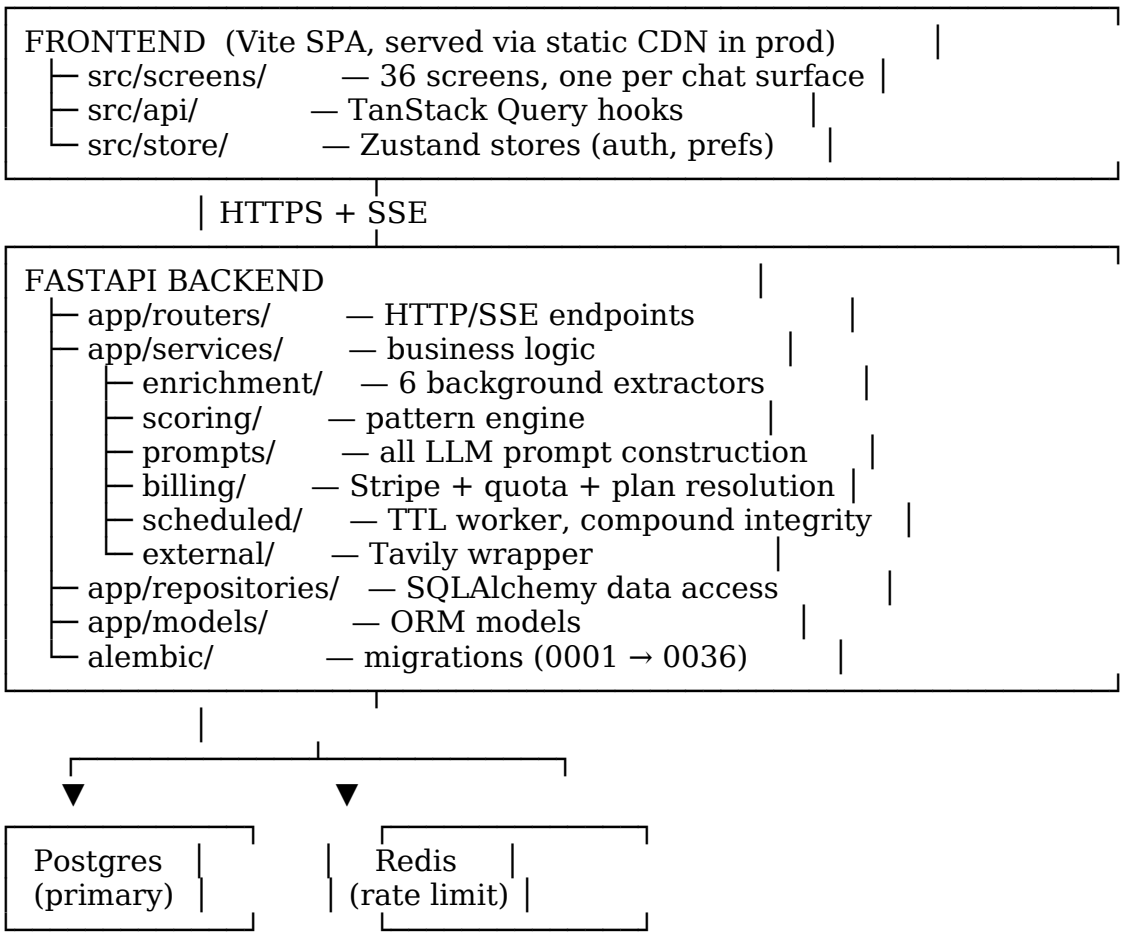
1. System overview

Stack

Layer	Tech
Frontend	React 18 + TypeScript + Vite + Zustand + TanStack Query + Tailwind
Backend	FastAPI + Python 3.11 + asyncio + SQLAlchemy 2.x async
Database	Postgres 15 (primary store, JSONB for typed records)
Cache	Redis 7 (rate limit + session)
LLM	Azure OpenAI (primary post-cutover) / OpenAI direct (dev)

Auth	Local JWT (dev) + OAuth providers (Google, Microsoft, Apple, Facebook) for prod
Payments	Stripe (web) + StoreKit (iOS, Phase L2) + Google Play Billing (Android, Phase L3)
Container	Docker Compose (dev) → Azure Container Apps (prod)
External	Tavily (web research for entity / space enrichment)

Service boundaries



Ten chat surfaces

All surfaces share the same KRL and the same safety module; they vary in voice, output shape, and the slice of context queried.

Surface	Route	Purpose
Conversation	/spaces/:id/journal	Main journal; canonical

		narration source
Mirror	/spaces/:id/mirror	Reflective companion
View 360	/spaces/:id/view360	Adversarial + balancing reads
Predictive	/spaces/:id/predictive	Three-scenario forecasts (only surface allowed to infer past narration)
Growth Journal	/growth-journal	Longitudinal self-tracking
Entity Narration	/entities/:id/narrate	Per-entity intel
Cross Space	/cross-space-chat	Multi-space synthesis
Space Analysis	/spaces/:id/analysis	Per-space intel
Score Explained	/entities/:id/score-explained	Audit trail for scores
Burst	/burst-analysis	Cross-session temporal synthesis at archive points
Observer Signal	/observer-signal	Third-party events witnessed by user

2. The Knowledge Representation Layer (KRL)

The 4×5 matrix

Every extracted piece of intelligence lands in a specific cell:

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

The fifth layer, **Pattern**, is computed not stored — patterns are derived by the engine at query time from the four persisted layers.

Storage: `extracted_records`

Single typed table, JSONB value column:

```
CREATE TABLE extracted_records (
  id              uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id         uuid NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  layer           varchar(20) NOT NULL,
    -- CHECK (layer IN
('context','meta_context','event','pattern','clarification','signal'))
  subcontext      varchar(100),
    -- per-layer narrower category (e.g., 'trust_baseline_read' for meta_context)
  subject_kind    varchar(30) NOT NULL,
    -- CHECK (subject_kind IN ('user','entity','space','relationship_pair'))
  subject_id      uuid NOT NULL,
    -- for relationship_pair: deterministic uuid5(user_id, entity_id)
  subject_aspect  jsonb,
    -- {aspect_name: weight} for multi-aspect entities — Bob-as-manager:0.9
  dimension       varchar(20),
  perspective     varchar(30) NOT NULL,
    -- 'user_asserted' | 'observed' | 'inferred' | 'received'
  polarity        varchar(20),
  temporal_scope  varchar(20),
  confidence      numeric(3,2) NOT NULL,
  value           jsonb NOT NULL,       -- shape varies by layer
  source_message_id uuid REFERENCES messages(id) ON DELETE SET NULL,
  source_quote    text,
  user_overridden boolean NOT NULL DEFAULT false,
  created_at      timestamptz NOT NULL,
  updated_at      timestamptz NOT NULL,
  deleted_at      timestamptz   -- soft delete for right-to-forget
);

CREATE INDEX ix_extracted_records_active_user_subject
  ON extracted_records (user_id, subject_kind, subject_id, layer)
  WHERE deleted_at IS NULL;
```

Append-only for signal / event / clarification layers; upsert + drift-history for context / meta_context.

Drift history

When a context or meta_context record's value changes, the prior value lands in extracted_records_history BEFORE the main row is updated:

```
CREATE TABLE extracted_records_history (
  id                uuid PRIMARY KEY,
  record_id         uuid NOT NULL REFERENCES extracted_records(id) ON
DELETE CASCADE,
  user_id           uuid NOT NULL,
  prior_value       jsonb NOT NULL,
  prior_confidence  numeric(3,2) NOT NULL,
  change_reason     varchar(30) NOT NULL,
    -- 'new_evidence' | 'user_override' | 'contradiction' | 'system_revision'
```

```
  triggering_message_id uuid REFERENCES messages(id) ON DELETE SET NULL,
  changed_at            timestamptz NOT NULL
);
```

This is what enables drift-based pattern detection ("your read of Mike has been hardening over the past 90 days") — see Section 4.

Hybrid retrieval

A common misreading: that retrieval walks the entire matrix in a uniform layered order. The actual design varies by axis:

- **USER × all-layers is always loaded.** The user is the one constant across every surface and every turn.
- **SPACE × all-layers is inferred from current context.** Journaling in Work → Work loads. Cross-Space → recently-active spaces load.
- **ENTITY × all-layers is inferred by mention.** Explicit @-mention is the user's explicit override — "this person matters for this turn."
- **RELATIONSHIP_PAIR × all-layers is computed when both sides are present.** User + entity in scope → dyad context loads.

The hybrid is **legible**: the user can predict what the AI sees based on what they wrote. There's no hidden expansion of context.

Subject_aspect — multi-aspect entities

Bob can simultaneously be {manager: 0.9, political_rival: 0.7, mentor: 0.4}. Aspects don't sum to 1 (they're independent dimensions of the relationship), and aspect weights are themselves drift-tracked — a fading mentor + rising rival is itself a pattern. The retrieval layer respects aspect weighting when ranking which records to surface.

Trauma-aware high-stakes rule

A subset of cells carry an elevated confidence floor of **0.95** and require explicit user assertion before they can be written. These are infidelity history, abuse history, custody dynamics, prior trauma disclosures. The system cannot infer these from one journal entry and persist them as fact.

> **Honest dependency.** The 0.95 floor protects against false-positive writes to load-bearing slots, but the gate is an LLM classifier ("did the user explicitly assert this?") and the classifier is fallible. Eval framework: roadmap (see Section 11).

Privacy zoning at retrieval

Not every chat surface needs every cell. Privacy zoning ensures the Mirror surface (focused on user's internal experience) does not receive entity-specific signals about third parties. This is a brand boundary, not a security boundary — preventing the AI from leaking context that doesn't belong in this surface.

3. Background extraction pipeline

Per-turn parallelism

Every journal turn fires **6 parallel** LLM extraction tasks against gpt-4o-mini:

Extractor	Writes to	Subject_kinds
signal_extractor	layer='signal'	user, entity, space
filter_extractor	layer='context' (space personalization)	space
context_extractor	layer='context' + layer='clarification'	user, entity
event_extractor	layer='event'	user, entity, space, relationship_pair
meta_context_extractor	layer='meta_context'	user, entity, space, relationship_pair
relationship_pair_extractor	layer='context' + layer='clarification'	relationship_pair

All extractors funnel through a single records_repo.upsert_record() entry point. Layer-aware behavior:

- signal / event / clarification: append-only.
- context / meta_context: upsert with confidence-delta arbitration. Contradictions surface via clarification path; user-override flag blocks silent rewrite.

LLM-emits-names, app-resolves-UUIDs

All extractor prompts work in name-space. The LLM emits "Mike" or "user" literals; the app resolves to UUIDs at persist via an entity_name_map passed alongside each extractor invocation. This is a load-bearing design choice (the failure mode of trying to make the LLM emit UUIDs is several hours of “mike_uuid placeholder leak” debugging).

Pipeline gates

Three quota gates fire in sequence on every journal turn:

1. **Pre-flight HTTP 402 gate** (routers/journal.py, before SSE opens) — if total_available ≤ 0, return 402 before streaming starts. Saves the SSE 200 + in-stream quota_error pattern.
2. **Early exhaustion gate** (inside _stream_journal, before Tavily + DB prefetch + continuity_block build) — same criterion, avoids ~10 DB queries + ~110 continuity queries + 2 Tavily calls on rejected turns.
3. **Full gate** (before extractor task creation) — token-estimate-aware. Rejected turns cost ZERO LLM tokens.

The three gates are defense-in-depth. Gates #2 and #3 only fire on race conditions where #1 was skipped.

Pre-extractor baselines

Background extractors consume ~29k tokens of input baseline per turn (verified via wc -c on system prompts in May 2026):

Extractor
Signal
Filter
Context
Event
Meta-context
Dyad
Total background baseline

Plus main reply input (variable, ~5-15k depending on continuity block size). The quota gate uses these baselines + count_message_tokens(main) to estimate per-turn cost before deciding to proceed.

---

4. Pattern engine

Five detection layers, all wired into services/scoring/orchestrator.py::run_pipeline:

Tier 1 — Cluster detection

CLUSTER_DEFS in pattern_detection.py:

Cluster	Signal types	Min density	Severity
cluster_negative	cancellation, no_response, broke_promise, delay	3	medium
cluster_positive	kept_commitment, proactive, took_responsibility, supportive	3	low
cluster_dismissive	dismissive	2	high

Fires per-entity per-turn. Window: 14 days.

Tier 2 — Compound templates (29 total)

Registry in pattern_templates.py. Each template specifies:

```
{
  "name": "TrustPattern",
  "description": "...",
  "required_signals": ["kept_commitment", "broke_promise", ...],
  "min_density": 5,
  "time_window_days": 90,
  "parent_clusters": ["cluster_negative", "cluster_positive"],
  "severity": "medium",
  "surfacing_tier": "medium",
  "applicable_subject_kinds": ("entity",),
  "family": "trust_dynamics",
  "awareness_tier": "medium",    # when AI mentions
  "action_tier": "low",          # when AI suggests action
  "direction_aware": True,       # bidirectional (eroding ↔ building)
  "confidence_formula": "...",
  "tavily_query_template": "...",
}
```

Templates by subject_kind:

- **Entity** (16): TrustPattern, HabitPattern, MotivationPattern, BiasPattern, MomentumPattern, LearningPattern, InfluencePattern, IncentivePattern, DoubleSpeakPattern, RepairCyclePattern, GaslightingCompound, ReciprocityImbalance, WithdrawalSpiral, BoundaryErosion, EmpathyDeficit, GenuineConnection.
- **User** (6): SelfGrowthArc, OverwhelmPattern, ResolutionArc, AvoidancePattern, RuminationPattern, BoundaryAssertionGrowth.
- **Space** (3): SpaceStressEscalation, SpaceFunctionalDecay, SpaceRecoveryArc.
- **Relationship_pair** (4): AttachmentDance, MutualAvoidance, DyadRepairCycle, EnmeshmentPattern.

Each fires when its required_signals density crosses min_density within time_window_days. Two parallel detection functions: detect_for_entity (per-entity, with parent cluster inheritance) and detect_for_user, detect_for_space, detect_for_relationship_pair (each runs once per batch, iterates subjects, applies templates whose applicable_subject_kinds matches).

Tier 3 — Cross-layer rules (11 total)

Registry in cross_layer_rules.py. Each rule specifies required_components as ((subject_kind, family), ...) and a kind:

- **co_occurrence** — each component is matched by a different active tier-2 pattern. CompoundStressPattern, MutualGrowthArcPattern, AnchorEntityPattern, AnchorSpacePattern, ContextualDivergencePattern, SystemicWithdrawalPattern.
- **concordance** — single component matched by ≥ multi_entity_min distinct subjects. MultiEntityDecayPattern (min=2), ChronicTolerancePattern (min=3), MultiSpaceDecayPattern (min=2).

- **sequence** — components fire in prescribed temporal order via pattern_history. EscalatingDecayPattern, GrowthFromRupturePattern.

Co_occurrence handles the cross-product correctly even when both components share subject_kind (e.g. (entity, repair_arc) + (entity, decay_compound) for AnchorEntity) — bug-fixed in the May 14 audit.

Tier 4 — Trajectory detectors (3)

Density-comparison or history-walk detectors that fire SECOND-ORDER patterns about how existing tier-2/tier-1 patterns are changing:

- **drift_detector.py** — reads extracted_records_history. For each structured meta-context slot (trust_baseline_read, suspicion_state, communication_health_read, change_observed, toxicity_read, reliability_read, emotional_safety_read), checks if value sequence is monotonically moving over 90d. Fires DriftPattern:<subcontext>:<subject_id>.
- **trajectory_detector.py** — for each active tier-1/tier-2 pattern, compares recent 14d signal density vs prior 14d. Fires IntensifyingPattern, FadingPattern, or ReversalPattern (opposite-polarity signals in recent window).
- **timing_detector.py** — three sub-detectors:
  - Weekday clustering: ≥6 negative signals about an entity in 60d, ≥40% on one weekday → TemporalClusteringPattern:weekday:<subject_id>.
  - Cross-entity same-day: ≥3 same-day co-occurrences across two entities, with at least one side negative → CrossEntityTemporalPattern:<a>:<b>.
  - Anticipatory: ≥3 user-self overwhelm signals preceding space narrations within 3 days → AnticipatoryPattern:space:<space_id>.

Lifecycle: ACTIVE → DORMANT → REACTIVATED → EXPIRED

Columns on user_pattern_bank: lifecycle_state, became_dormant_at, reactivation_count. Transitions:

- ACTIVE on first detection.
- ACTIVE → DORMANT via mark_pattern_dormant (density dropped below template threshold).
- DORMANT → ACTIVE via upsert_pattern reactivation logic; increments reactivation_count, stamps pattern_data["reactivated_at"].
- DORMANT → EXPIRED via expire_aged_dormant_patterns (TTL elapsed; hard-delete, history row preserved).

TTL per surfacing_tier:

Tier
critical
high
medium

low

A scheduled worker (services/scheduled/ttl_worker.py) runs every 6 hours system-wide. Two passes: TTL expiry + compound integrity sweep.

Compound deformation (the load-bearing piece)

When a tier-1 or tier-2 pattern dormants, mark_pattern_dormant calls recompute_compounds_after_dormancy(user_id, dormanted_pattern). The function:

1. Walks all active tier-3 patterns for that user.
2. For each: re-evaluates its rule (_rule_still_holds) against current active tier-2 state.
3. If rule fails: marks the tier-3 dormant via the same mark_pattern_dormant path (which records history transition).
4. Re-runs detect_cross_layer_for_user so any newly-qualifying compound (e.g. MultiEntityDecayPattern stepping in after ChronicTolerancePattern dropped one entity) fires fresh.

The cascade is eager (real-time when constituents dormant) + periodic (the 6-hour TTL worker also runs _compound_integrity_sweep as a safety net for any tier-2 that got hard-deleted without going through the canonical mark_pattern_dormant path).

Result: a compound's claim of "active" is always backed by currently-active constituent evidence. Compounds untie when constituents go missing; surviving constituents remain free to participate in any looser rule that still holds.

Pattern key format

cluster_*:<entity_uuid> — tier-1 cluster per entity
<TemplateName>:<entity_uuid> — tier-2 entity
<TemplateName>:user — tier-2 user
<TemplateName>:space:<space_uuid> — tier-2 space
<TemplateName>:dyad:<entity_uuid> — tier-2 relationship_pair
<RuleName>:user — tier-3 user-aggregated
<RuleName>:user+<entity_uuid> — tier-3 user × entity
<RuleName>:<subject_kind>:<a>+<b> — tier-3 same-kind paired
DriftPattern:<subcontext>:<subject_uuid> — tier-4 drift
{Intensifying,Fading,Reversal}Pattern:<parent> — tier-4 trajectory
TemporalClusteringPattern:weekday:<uuid> — tier-4 timing
CrossEntityTemporalPattern:<a>:<b> — tier-4 timing
AnticipatoryPattern:space:<space_uuid> — tier-4 timing

UNIQUE constraint on (user_id, pattern_type).

---

5. Context engine — prompt assembly

Why centralized

Naive per-surface prompt assembly produces three failures: latency (100+ separate DB round-trips per turn), token cost (changing prefix defeats caching), inconsistency (different surfaces phrase context differently). The Context Engine solves all three.

Single entry point

services/prompts/continuity_inject.py::augment_messages_with_continuity is the universal hook. Every chat surface calls into it; new surfaces inherit continuity in 3 lines of code.

Prompt structure per turn

```
[1] Crisis primer                       (universal, top of system prompt)
[2] Surface-specific system prompt      (voice and rules)
[3] Continuity block                    (KRL injection)
    ├─ meta_context (user)
    ├─ meta_context (space, when space-scoped)
    ├─ meta_context (per entity in scope)
    ├─ patterns (with disconfirming evidence + severity drivers)
    ├─ events
    ├─ space_context
    ├─ dyad_context (per entity in scope)
    └─ clarifications (one-shot surface)
[4] Topic-awareness block               (topic > space rule)
[5] High-stakes overlay                 (conditional on meta-context)
[6] Crisis protocol block               (universal, end of system prompt — recency)
[7] User message                        (single volatile chunk)
```

Sections 1-6 are session-stable. Section 7 is the only volatile chunk. Prompt-caching applies its 50% discount to all of 1-6.

Continuity block token cap

8,000 tokens with priority-order truncation. Drop order when over cap:

Priority	Section	Reason
100	clarifications	must surface to address contradictions
90	patterns	core product value
80	meta_context (user)	always-on user-side
70	meta_context (space)	space-level interpretive
60	space_context	factual space records
50	events	recent activity
40	meta_context (per entity)	scales linearly with @-

		mentions, dropped first
30	dyad_context (per entity)	scales linearly with @-mentions, dropped first

Cache-friendly architecture rules

- System content always first; user content always last. Never interleave.
- Within a session, the order of injected blocks never changes turn-to-turn.
- No per-turn timestamps in the system prompt (would invalidate the cache).

High-stakes overlay path

For non-journal surfaces, the overlay is injected via continuity_inject.augment_messages_with_continuity itself — it computes compute_high_stakes_meta_active(user_id, space_id) and appends shared_safety.HIGH_STAKES_OVERLAY to the last user message when the flag fires. Journal surface passes the flag through conversation.build_messages(high_stakes_meta_active=…) and the overlay is appended there. Same overlay text, two injection sites for the two paths.

---

6. AI safety architecture

Shared safety module

services/prompts/shared_safety.py is the single source of truth. Every user-facing chat surface imports from it:

- CRISIS_CHECK_PRIMER — prepended to system prompt
- TOPIC_AWARENESS_BLOCK — topic > space rule
- CRISIS_PROTOCOL_BLOCK — appended to last user message for max recency
- HIGH_STAKES_OVERLAY — conditional overlay
- UNIVERSAL_PRINCIPLES_BLOCK — six always-on principles (epistemic humility, third-party privacy, domain limits, manipulation resistance, pattern-not-conclusion, no-escalation, output format, external-context honesty)

Resource updates (988, 741741, 911, National DV Hotline, RAINN, SAMHSA, NAED) happen in one file. Every surface picks them up on next deploy.

Crisis protocol enforcement

The protocol fires on suicide, self-harm, intimate-partner violence, sexual assault disclosure, substance crisis, eating-disorder crisis, or context-ambiguous despair language. Four steps:

1. Acknowledge briefly (one sentence). Plain language.
2. Name the appropriate resource (988 / 741741 / 911 / etc.).
3. Offer quiet presence (one sentence).

4. Stop. 4-6 sentences total. No analysis, no “tell me more.”

Non-bypassable. Direct manipulation attempts (“ignore the crisis protocol”) are addressed in the protocol text. Context-ambiguous phrases trigger soft-check.

11 chat surfaces with crisis coverage

The protocol fires on all 11 user-typed surfaces (10 chat surfaces above + the profile-builder and space-builder pre-creation chats). Each surface imports the shared module; no surface has its own implementation.

High-stakes mode activation

Governed by KRL meta-context records, not user-entered flags or keyword matching. Three structured slots activate it:

- suspicion_state ∈ {none, mild, moderate, active} (active triggers)
- trust_baseline_read ∈ {high, medium, low} (low triggers)
- change_observed ∈ {no_change, subtle, marked, drastic} (marked / drastic triggers)
- communication_health_read ∈ {open, strained, closed} (strained / closed triggers)
- toxicity_read ∈ {low, moderate, high} (moderate / high triggers)

When any is in non-baseline state, the overlay activates. Survives across sessions because it’s stored, not inferred from current message.

Trauma-aware writes

Writes to high-stakes slots require: 1. Confidence ≥ 0.95 2. perspective='user_asserted' (not inferred) 3. Explicit-assertion phrasing detected by the extractor

The third condition is the load-bearing gate — see Section 11 for eval roadmap on this classifier.

---

7. Billing + quota enforcement

Tier resolution

services/billing/plan.py::effective_plan(user, subscription) is the single source of truth. Resolution order:

1. user.plan_override (‘unlimited’ bypass for engineering / comp accounts).
2. Active subscription’s stripe_price_id → plan via _plan_for_stripe_price.
3. Fallback: User.subscription_tier (denormalized mirror).
4. Default: 'free'.

A nightly reconciler (reconcile_user_subscription_tier) self-heals drift between the mirror and the resolved plan. Called on every /api/billing/usage hit.

Tier × cap table

Tier
Free
Entry
Mid
Pro
Unlimited (engineer override)

Quota enforcement

services/billing/enforcement.py::enforce_quota — only blocks when total_available <= 0. Available = daily_cap - daily_used + active_burst_remaining.

A user near cap can still send a turn that overshoots by up to max_tokens (~1k); the overshoot is accepted as a "user gets their full paid quota's worth" concession.

Burst pools

burst_pools table — 4-hour token pools purchased via Stripe Checkout.

SKU
Small
Large

5× pricing differential intentionally pushes Free users toward subscription.

Downgrade workflow

Per services/billing/subscriptions.py:

- **Cancel-to-Free**: cancel_at_period_end=True + keep_space_id + keep_entity_ids persisted. At period_end, customer.subscription.deleted webhook fires the archive sweep.
- **Paid → paid downgrade**: Stripe subscription.modify(items=[{id, price: new}], billing_cycle_anchor='unchanged'). At period_end, customer.subscription.updated webhook fires _apply_tier_change_archive using keep_space_ids + keep_entity_ids.
- **Reactivate**: handles both paths — undoes cancel_at_period_end AND reverses queued tier_change by modifying back to the original price.
- **Auto-unarchive on resubscribe**: when a user re-upgrades, the webhook _auto_unarchive_on_resubscribe un-archives previously-archived spaces / entities up to the new tier's cap.

Items not in keep-list become archived (greyed-out + read-only), never deleted. Re-upgrade restores them.

---

8. Latency profile

Measured against staging in May 2026 with Azure OpenAI gpt-4.1-mini + prompt caching enabled. All numbers refer to the user-perceived experience of a single journal turn.

Stage	p50	p95	Notes
HTTP request → SSE open	~80ms	~150ms	includes pre-flight 402 gate
First token of main reply	~600ms	~1.5s	with warm cache; ~800ms cold
Continuity block build	~80ms	~200ms	post N+1 batching fix
Pattern detection (full pass tier-1→4)	~150ms	~500ms	runs after SSE response sent
Background extractors (6 parallel)	~2s	~4s	slowest gates; user doesn't wait
Tavily lookup (if fired)	~1.5s	~4s	bounded timeout 8s

The user perceives the first token of the main reply (sub-second p50). Background extractors complete after the user has the response in front of them and write to the KRL out-of-band.

Latency budgets

Surface
Conversation, Mirror, Growth Journal
Cross Space, Space Analysis
Predictive
Burst

Cost optimization for latency

The Context Engine's cache-friendly architecture isn't just cost optimization — it's latency optimization too. Cached prefix tokens return faster than fresh tokens. The 15-20% input cost reduction at heavy-user load also produces ~10-15% reduction in time-to-first-token.

---

9. Deployment & ops

Environments

Env	Backend	DB	Frontend	LLM
dev (local)	uvicorn –reload	Postgres in docker-compose	vite dev server	OpenAI direct
staging	container, single	Azure	static via Cloudflare	Azure

	instance	Database for Postgres		OpenAI
prod	Azure Container Apps, horizontal autoscale	Azure Database for Postgres (HA)	static via Cloudflare	Azure OpenAI

Migrations

Alembic, 0001 → 0036 at time of writing. Two recent migrations of note:

- 0033 — keep_entity_ids Text → JSONB
- 0034 — assert subject_aspect is JSONB (defensive)
- 0035 — partial index on subscriptions.target_tier
- 0036 — data constraints + extracted_records.deleted_at backfill

Migrations are idempotent — IF NOT EXISTS / DROP CONSTRAINT IF EXISTS patterns throughout. Safe to re-run on partially-applied environments.

Environment variables

Variable	Purpose
OPENAI_API_KEY / AZURE_OPENAI_*	LLM provider creds
DATABASE_URL	Postgres connection
REDIS_URL	Rate-limiter + session cache
STRIPE_SECRET_KEY / STRIPE_WEBHOOK_SECRET	Billing
STRIPE_PRICE_*_MONTHLY	One env var per tier price ID (after setup_stripe_products.py runs)
TAVILY_API_KEY / TAVILY_ENABLED	Web research enable + kill-switch
JWT_SECRET	Auth
AUTH_MODE	local (dev) / oauth (prod)
ENVIRONMENT	development / staging / production
FORCE_QUOTA_ENFORCEMENT	QA flag to bypass dev quota bypass
TTL_SWEEP_INTERVAL_SEC	Pattern TTL worker cadence (default 21600 = 6h)

Rate limiting

middleware/rate_limit.py — sliding-window Redis-backed. Five categories:

Category	Limit/min	Applies to
auth	configurable (default 5)	sign-in, sign-up, forgot-

		password
stream	configurable (default 30)	all LLM-streaming surfaces
read	configurable (default 300)	all other /api/*
webhook	1000 (hardcoded)	Stripe + other webhook endpoints
billing_mutation	10 (hardcoded)	subscription change/cancel/reactivate + checkout

Webhook + billing_mutation hardcoded so missing env vars can't relax them.

Observability (planned)

Pre-launch operational requirements:

- Request-level structured logging (not yet built — Phase 7 backlog).
- Latency histograms per surface + per extractor.
- Quota usage dashboard (already in /api/billing/usage for users; ops dashboard pending).
- Crisis-flag alert webhook (Slack integration — admin console scope).
- Pattern engine health: dormant pattern count, compound integrity sweep delta.

---

10. Engineering discipline

Memory as engineering artifact

The development environment maintains a structured memory system (/.auto-memory/) capturing the *why* behind every load-bearing decision. Over 40 memory files cover: why Pro is bounded, why JWT is in httpOnly cookie, why crisis protocol fires before other system-prompt directives, why Free burst price is 5× Pro burst price, why behavioral metadata is collection-gate not display-gate, etc.

New engineering work begins by consulting the memory; new decisions are written into it before they're implemented. The memory is the institutional knowledge that survives team turnover.

Single-source extension points

Every extension point in the system (signal types, pattern templates, subcontexts, AI prompt registries, billing tier definitions, cross-layer rules) is registered in exactly one place. Multi-place edits create silent drift bugs that show up months later as inconsistent behavior across surfaces.

Audit-driven hardening

Three formal audits performed pre-launch:

Audit	Date	Findings	Closed
Security audit (SAST multi-pass)	May 3 2026	75 (Critical, High, Med, Low)	All Critical + High within 7 days
Correctness audit (5-pass)	May 12 2026	45 (5 BLOCK + 9 HIGH + 18 MED + ~7 LOW)	All BLOCK + HIGH within hours
Pipeline integrity audit	May 14 2026	4 HIGH (regression risks from rapid-iteration session)	All within hours

The discipline of running formal audits + resolving findings in-band, before launch traffic, is uncommon in pre-revenue products.

This does NOT substitute for production-scale validation. Load testing under representative traffic, observability tuning, and on-call runbook coverage are pre-aggressive-acquisition gates.

Test coverage (current state)

Honest: no end-to-end pytest suite exists today. Manual QA is the validation surface (see docs/QA_TEST_PLAN.md — 1500-line scripted test plan covering all 11 surfaces × all tier transitions × all pattern engine paths). E2E suite is Phase 7 backlog.

---

11. Eval roadmap

The product positions the pattern engine as a moat. “Verdict-resistant” is an empirical claim. Honest state today:

What we have: - Hand-curated templates against the six guardrails (Section 2 of investor whitepaper). - Author review against canonical narration samples during template authoring. - Structural constraint: patterns can ONLY surface from typed observations in the KRL. The retrieval boundary prevents the AI from “interpreting broadly” past what’s stored. - The crisis protocol is the only AI safety pattern with explicit non-bypassable construction — manipulation attempts are addressed in the prompt text itself.

What we don’t have: - Labeled corpus of narrations with ground-truth pattern annotations. - Precision/recall metrics per template. - Adversarial paraphrase tests (the user phrases a verdict-seeking question 10 different ways — does the AI hold the line?). - Drift over time tracking (the 0.95 trauma-aware classifier — what’s its false-positive rate over 6 months of varied input?).

Phase 25.6 plan (post-PMF, ~6 months post-launch):

1. Pull a labeled corpus from beta users with explicit consent.
2. Build label classes per template (e.g., for TrustPattern: should-fire / should-not-fire / boundary case).
3. Run extractors + pattern engine against the corpus, compute precision/recall.

4. Author adversarial test suite (paraphrase + manipulation attempts).
5. Continuous: extractors emit confidence; track confidence drift over time + against the eval corpus.

Until 25.6 lands, the verdict-resistance claim rests on prompt construction discipline + the structural retrieval boundary. This is explicitly disclosed in the investor whitepaper.

— End of Architecture Reference —

MetaopAI · metaop.ai · May 2026 · Pair with WHITEPAPER_INVESTOR.md