Ambient Knowledge — Product & Architecture Write-up

Timebox: built as a 4–8 hour take-home MVP.

TL;DR

Ambient Knowledge is an **AI collaborator for workplace communication** that lives *inside the message composer*.

It proactively finds relevant internal context while you write, shows **grounded sources**, and lets you attach or link that context with one click — reducing back-and-forth without turning the product into "yet another chatbot."

1) Product Vision

Workplace context is fragmented across docs, tickets, incident notes, and prior threads. As a result:

- Senders spend time searching (or skip it) and messages go out missing key background.
- Recipients ask clarifying questions, causing delays and context-switching.

Ambient Knowledge makes the *default* action "send a better message" by making context retrieval **automatic, calm, and auditable**.

2) Thesis (how AI should facilitate collaboration)

AI should act like **invisible infrastructure** that reduces coordination cost:

- Prefer **fewer steps and fewer screens**, not new workflows.
- Make improvements **legible** (why this context? what sources?) so teams can trust it.
- Optimize for **shared understanding** between people (sender + recipient), not just helping one user type faster.

This MVP fits that thesis by putting AI in the smallest useful surface: the composer and the message view — a collaborator that helps you include what your teammate will need.

3) What This Is (and isn't)

Ambient Knowledge surfaces relevant organizational context while a person writes or reads a message.

- It **is** an inline collaborator: optional, grounded in real artifacts, and designed for one-click usefulness.
- It **is not** a chatbot UI or a separate prompt destination.

4) How to run

From a shell, set `REPO_ROOT` to the folder that contains this repo, then:

```
export REPO_ROOT="/path/to/your/code"
cd "$REPO_ROOT/ambientKnowledge"
cd impl
npm install
npm run dev
```

Optional Claude synthesis:

- Copy `impl/.env.example` → `impl/.env.local`
- Set `ANTHROPIC_API_KEY=...`

Note: AI synthesis requires a valid key. If the key is missing/invalid, `/api/context` will error.

---

5) Design Principles

- **AI as infrastructure, not destination.** The best AI interaction is invisible. Remove workflow steps rather than creating new surfaces.
- **Calm by default.** Never interrupt typing. Suggestions are dismissible and optional. Low confidence → show less.
- **Grounded and auditable.** Every summary is attributable to retrieved sources. No hallucination — generation is constrained to retrieved snippets.
- **Operational controls.** Under load shedding or rate limiting, the system can choose retrieval-only instead of synthesis.
- **Cost-aware intelligence.** Use expensive inference only when justified. Prefer deterministic retrieval and cached summaries first.

---

6) Core User Journeys

Adding context while composing: User drafts a message → system retrieves related artifacts → proposes a context card (topic + summary + source links) → user attaches with one click → message sends with compact context block.

Understanding an incoming message: User receives a message with unfamiliar references → taps "What's this about?" → system retrieves and synthesizes background → user sees summary + source links inline.

---

7) Engineering Decisions

This section exists to answer “why this stack/architecture, and what did you consider instead?”

Stack choices

- **Next.js (App Router) + TypeScript:** fast iteration, server routes for retrieval/synthesis, strong typing for extensibility.
- **Tailwind CSS:** speed + consistency for a polished UI within the timebox.
- **Framer Motion:** small, intentional transitions to make AI suggestions feel calm instead of jumpy.

Architecture choices

- **Tiered serving (cache → retrieval-only → synthesis):** keeps the composer responsive and cost predictable.
- **Retrieval-first design with sources:** maximizes trust and debuggability; LLM is a constrained layer on top.

Alternatives considered (not implemented in MVP)

- Embeddings + vector DB (pgvector/Pinecone) and BM25 hybrid search.
- Streaming synthesis and async enrichment via background workers/queues.
- Redis-backed cache/rate limits for multi-instance deployments.

---

8) Technical Stack

Layer	Choice
Framework	Next.js 15 App Router, React, TypeScript
Styling	Tailwind CSS + Framer Motion
API	`/api/context` route (retrieval + synthesis)
Retrieval	Glean-inspired local ingestion + chunk index (hybrid lexical + lightweight semantic proxy)
Synthesis	Anthropic Claude
Data	26 structured mock knowledge artifacts + 11 seeded channels/threads

---

9) What's Implemented

Context Engine (`contextEngine.ts`)

Three-tier serving pipeline that routes every request from cheapest to most expensive:

Tier	Mechanism	Latency
Cache	Reuse prior context for similar intent + channel	~10ms
Retrieval-only	Deterministic summary from top snippets, no LLM	~50ms
Synthesis	Full LLM generation from retrieved context	~1–3s

Supporting infrastructure:

- **In-memory cache** with 45s TTL, 5000 entry max, freshness metadata.
- **Request deduplication** — identical in-flight queries share one response.
- **Rate limiting** — fixed-window (20 req/10s per client).
- **Load shedding** — max 24 concurrent synthesis calls; excess downgrades to retrieval.
- **Mode-aware scoring** — `compose` mode biases toward recipient relevance; `incoming_lookup` mode biases toward message topic.

Retrieval Engine (`retrieval.ts`)

Glean-inspired retrieval flow:

- **Ingest** knowledge items and normalize fields.
- **Chunk** bodies into paragraph/sentence-ish segments.
- **Hybrid score** chunks (lexical TF-IDF-ish + deterministic semantic proxy) and **collapse** best chunks back to document-level hits.
- **ACL filter** by viewer principals (demo: `group:all` + current user).

This is implemented in `gleanIndex.ts` and called by `retrieval.ts`, while preserving the same `retrieve()` interface used by the context engine.

Knowledge Base (26 artifacts)

Covers multiple project areas (platform, mobile, design system, onboarding, security/ops, release) to demonstrate retrieval differentiation:

Domain	Example Artifacts
Platform Migration	Monolith decomposition, billing DB dual-write strategy
Mobile App v2	React Native rewrite, transaction list performance
API Gateway	Kong vs Envoy benchmark, rollout release notes
Design System	Cross-platform component library, Figma token pipeline
Customer Onboarding	Guided setup wizard, A/B test results (34% activation lift)
Security & Ops	Q1 audit findings (IDOR, JWT), Feb 18 incident postmortem

Each channel surfaces a distinct top artifact — billing migration, design system tokens, onboarding metrics, security findings, gateway release notes, RN performance.

Channel & DM Navigation

- 11 channels (including incident/security/release/QA/leadership) + multiple DM threads.
- Discriminated union routing (`ActiveView = channel | dm`) with independent message stores.
- Sidebar highlights correct selection for both channels and DMs.

Browser History & State Preservation

- `history.pushState()` with serialized `NavState` (view, attached context, draft text) on every navigation.
- `replaceState()` snapshots current state before leaving, so mid-composition drafts survive.
- Back/forward restores the full view, attached context, and draft text. The context suggestion hook re-fires immediately on restore.

Context Lookup Gating

Heuristic determines when to show the "What's this about?" button on incoming messages:

- **Suppress** for simple replies/acknowledgements ("Perfect, got it, I'll do X").
- **Show** for messages with clarification signals (questions, approvals, decisions) or technical content (specific terms, metrics, version refs).

Wiki Pages

Dynamic `/wiki/[id]` route renders any knowledge artifact as a full page with related items. Source links in context cards link directly to these pages.

---

10) Tradeoffs (and what I'd do with more time)

Tradeoffs made for speed and clarity in a 4–8 hour MVP:

- **Mock connectors/data:** knowledge comes from a seeded JSON store instead of real integrations.
- **No real auth/ACL graph:** permission checks are simplified to keep the demo coherent.
- **Deterministic "semantic proxy":** avoids embedding infra; keeps the demo reproducible offline.
- **In-memory caching/rate limits:** good for the prototype; not production-grade multi-instance.

With more time:

- Replace semantic proxy with real embeddings + hybrid retrieval, preserve citations.
- Add async enrichment + streaming so retrieval appears immediately and synthesis upgrades in-place.
- Add Redis + queues for true multi-tenant scaling and fairness.
- Expand “incoming message understanding” into a first-class workflow (reply suggestions, shared context snippets).

---

11) Scaling to 1000+ Users

At organizational scale, "LLM per keystroke" fails. The core risks:

- **Request volume explosion** — typing-triggered requests × concurrent users = high QPS spikes.
- **LLM saturation** — throughput, latency, and cost destabilize during peaks.
- **Retrieval fan-out** — slow sources block the whole suggestion pipeline.
- **Multi-tenant fairness** — heavy teams starve others without controls.

Architecture Strategy

Tiered inference (already implemented): Cache → retrieval-only → synthesis. The agent is never the bottleneck for baseline usefulness.

Admission control: Per-user/workspace token budgets, QPS caps, and queue depth thresholds. Downgrade to lower tier rather than block.

Async enrichment (planned): Return retrieval result immediately; upgrade with synthesis when ready. User sees context improve in-place.

Hybrid retrieval (partially implemented): The demo already uses lexical + a lightweight semantic proxy with ACL filtering. At scale, you’d replace the semantic proxy with real embeddings + vector search (and likely keep BM25 in the mix).

AmbientKnowledge vs Glean (explicitly)

What this demo borrows from Glean’s *shape*:

- **Connector → ingest → index → retrieve → synthesize** mental model (even though the demo’s “connector” is a JSON seed).
- **Chunk-level retrieval** with doc-level collapsing.
- **Permission filtering** (ACL principals) applied during retrieval.
- **Hybrid scoring** (lexical + semantic signal) rather than pure keyword.

What Glean does that this demo intentionally does not:

- **Real connectors + incremental crawls** (Slack/Drive/Jira/GitHub/etc), change detection, retries.
- **True embeddings + vector infrastructure** (model serving, vector DB, ANN search tuning).
- **Enterprise ACL graph** (groups, nested groups, external shares, doc-level vs field-level permissions).
- **Personalization at scale** (click feedback loops, recency/affinity graphs, per-user ranking).
- **Hard multi-tenant isolation** (per-tenant indexes, quotas, compliance controls).

Context precomputation: Background jobs pre-generate context bundles for high-traffic channels and recurring topics.

Model routing: Smaller models for first-pass synthesis; escalate to larger models for low-confidence/high-importance cases.

UX Under Load

Users never experience a blocked composer. Under constrained conditions: show deterministic context quickly, mark confidence clearly, allow immediate attach/send, upgrade opportunistically when richer synthesis arrives.

Target SLOs

Tier	p95 Latency	Availability
Cache / Retrieval	< 700ms	99.9%
Synthesis (async)	< 2.5s	99.5%

---

12) API Contract

Request: `{ draftText, recipientId, channelId?, mode?: "compose" | "incoming_lookup" }`

Response: `{ topic?, summary?, openQuestions?, sources?, confidence?, servingTier?, freshnessMs?, debug? }`

---

13) What's Next

Priority	Improvement
1	Replace in-memory cache/rate-limit with Redis for multi-instance support
2	Async synthesis enrichment — return retrieval immediately, stream upgrade
3	Production hybrid retrieval (BM25 + embeddings) with permission filtering
4	Per-tenant fairness quotas and priority classes
5	Observability dashboards (tier mix, cache hit rate, queue depth, p95 latency)

---

14) Success Metrics

Product: Context attach rate · "What's this about?" usage · Reduction in follow-up clarification messages · Time-to-send improvement.

System: p95 latency by tier · Cache hit rate · Synthesis fallback rate · Cost per context card.

---

15) How this meets the evaluation criteria

- **Extensibility/readability:** clear separation of retrieval, synthesis, and UI; typed contracts and small modules.
- **Creative AI-human collaboration:** AI helps people send *better, more complete* messages by attaching shared context (not just "help me write").
- **Polish & UX:** calm, optional suggestions; one-click attach/insert; sources always visible.

- **Beautiful simplicity:** AI reduces screens (no chatbot surface) and reduces work (automatic retrieval).
- **Robustness:** tiered serving + load shedding makes failure modes graceful and cost-aware.

---

16) Graceful Degradation

If LLM synthesis is unavailable (missing key, rate limiting, transient errors), the system downgrades to **retrieval-only** instead of blocking the composer. The user still gets relevant context with visible sources.