Ambient Knowledge — Write-up (main + appendix)

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

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

Instead of asking users to "go chat with AI," it uses Claude as the collaborator model to **turn internal artifacts into a compact context card** you can attach or link with one click.

Main write-up (3 pages max)

1) Product vision — what I built and why

Workplace context is scattered across docs, tickets, incident notes, and old threads. Messages often go out without enough background, which creates:

- time lost searching (or skipping the search),
- follow-up questions and delays,
- misalignment when decisions are made on partial context.

What I built: a Slack-like messaging UI where the system surfaces relevant internal artifacts while you draft a message. The UI keeps AI calm and optional: you can attach the context, insert a link, or dismiss it.

The goal is not "write for me," it's "help me send something my teammate can act on."

2) Thesis — how AI should facilitate collaboration

AI should reduce coordination cost by acting like **infrastructure**:

- **Remove steps** (no new screens, no prompt rituals).
- **Support shared understanding** between sender and recipient (context + sources, not just prose).
- **Be auditable** (show why something was suggested and where it came from).

This MVP embodies that thesis by putting AI in the smallest useful surface: the composer and the message view.

3) UX — the core loop

1. Pick a channel or DM and start typing.
2. A context card appears when relevance is high: **topic + 1–2 sentence summary + sources**.
3. One-click actions:
   - **Attach**: includes a compact context block with the message.
   - **Insert link**: drops a `/wiki/...` source link into the draft.
   - **Dismiss**: keeps the composer calm.
4. Clicking sources opens the underlying wiki page for verification.

4) Engineering decisions — stack, architecture, alternatives

Stack

- **Next.js (App Router) + TypeScript**: fast iteration, clean server boundary for `/api/context`, strong typing for extensibility.

- **Tailwind + Framer Motion**: polish within the timebox; subtle transitions make suggestions feel calm.

Architecture

- **Retrieval-first with citations**: trustable by default; Claude is constrained to retrieved snippets.
- **Caching + request dedupe + rate limiting**: keeps latency predictable and prevents runaway costs.

Alternatives considered (not built in MVP)

- Embeddings + vector DB + hybrid search.
- Streaming/async enrichment via background workers and queues.
- Redis-backed cache/rate limiting for multi-instance deployments.

5) Tradeoffs (timebox) + what I'd do with more time

Tradeoffs to ship a cohesive end-to-end experience quickly:

- **Mock knowledge base** instead of real connectors.
- **Simplified auth/permissions** (enough to demonstrate the shape, not enterprise-grade).
- **Deterministic semantic proxy** rather than embedding infrastructure.

If this were a real product seed:

- Replace the semantic proxy with embeddings + hybrid retrieval while preserving citations.
- Add streaming synthesis + background enrichment.
- Add Redis + a queue for fairness, retries, and multi-tenant scaling.

6) How to run

```
cd humans&/ambientKnowledge/impl
npm install
npm run dev
```

Set `ANTHROPIC_API_KEY` in `humans&/ambientKnowledge/impl/.env.local`.

7) Evaluation alignment

- **Extensibility/readability:** clear separation between retrieval, synthesis, and UI.
- **Creative AI-human collaboration:** AI helps people share context, not just generate text.
- **Polish & UX:** calm defaults, one-click usefulness, source transparency.
- **Beautiful simplicity:** fewer screens and fewer steps because the intelligence does the searching.

Appendix (technical deep dive)

A1) System shape (end-to-end)

Ingest → index → retrieve → synthesize → attach/link

- Knowledge artifacts are seeded from a JSON store.
- The UI calls `/api/context` as the draft evolves.
- The server retrieves relevant snippets and calls Claude to produce a compact, grounded card.

A2) Context engine (serving + controls)

Key mechanisms used to keep the system understandable and scalable:

- **Cache**: reuse recent results for similar intent.
- **Request deduplication**: identical in-flight prompts share work.
- **Rate limiting**: caps bursts from a single client.
- **Load shedding**: bounds concurrent synthesis calls.

A3) Retrieval approach (Glean-inspired, simplified)

- Normalize knowledge items and chunk bodies.
- Score chunks with a hybrid signal (lexical + deterministic semantic proxy).
- Collapse chunk hits to document-level results.
- Return sources with URLs so users can verify context.

A4) Knowledge base coverage

Seed data covers multiple domains to make retrieval behavior legible:

- platform migration + billing
- API gateway rollout + release notes
- security audit + incident postmortems
- design system tokens
- onboarding experiments

A5) Scaling to 1000+ users (what would change)

At organizational scale, keystroke-triggered AI can overload both retrieval and synthesis. A production plan:

- async enrichment (return fast results, upgrade when ready)
- hybrid retrieval (BM25 + embeddings + permissions)
- shared infra (Redis, queues)
- observability (tier mix, p95 latency, cost per suggestion)

A6) API contract

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

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