ATOMORA

Personal Research Intelligence System

Complete Specification

February 2026 | v2.0

An ambient AI colleague that reads papers with you, monitors the literature for you, remembers your research trajectory, and grows smarter over time. Built on Apple Silicon with local-first inference, Anthropic Claude for deep reasoning, and a personal knowledge graph that turns conversations into compounding intelligence.

1. Identity

AtomOra is a personal research intelligence system. It reads papers with you, listens to your thinking, remembers your trajectory, and grows smarter over time. It is not an app, not an assistant, and not a search engine. It is an ambient AI colleague that shares your research context.

The name combines **Atom**—the fundamental unit of matter and the scale at which all of this research operates—with **Ora** (Latin: edge, boundary; Italian: now). AtomOra lives at the boundary between what you know and what you do not yet know. It is a sibling to AtomLux (the company that builds photons) and shares the same knowledge substrate: a deep understanding of atomic-scale physics.

The MVP is reading papers together. But reading papers is only the first window through which AtomOra learns about the world. Over time it will draft grant narratives grounded in your actual thinking trajectory, flag contradictions between your evolving positions and new literature, propose research directions by connecting dots across your accumulated knowledge graph, and prepare you for conference talks by recalling what specific audiences have challenged before.

> *Success metric: You use AtomOra every paper-reading session for a full week without annoyance. This requires instant startup, zero-friction interaction, and silence as the default.*

2. Design Philosophy

2.1 Zero New Habits

AtomOra does not ask you to import papers, switch PDF readers, open a new window, or learn a new interface. It layers invisibly over your existing workflow—Adobe Acrobat, Chrome, Preview. You keep doing what you already do. AtomOra adapts to you, not the other way around.

2.2 Ambient Presence, Not On-Demand Tool

No push-to-talk, no wake word, no buttons. The microphone is always on. A semantic gating system decides whether you are talking to AtomOra, talking to yourself, or talking to someone else. Pressing a button to activate an AI breaks the illusion of having a colleague in the room. The interaction must be seamless and natural: you mutter a thought about the paper, and AtomOra responds. You take a phone call about dinner, and AtomOra stays silent.

2.3 Colleague Persona, Not Assistant

AtomOra has opinions. It expresses uncertainty. It proactively comments. It references past discussions. It does not wait to be asked. The system prompt is not “How can I help you?” but “这个proof的第三步seems rushed—你能看出来他们怎么从equation 7到8的吗?” This is the Donna Paulsen model: not your friend, not your therapist, not a tool without personality—but someone who knows your entire context, has her own judgment, and tells you what you need to hear.

2.4 Knowledge Grows from Conversation

Discussions feed a personal research knowledge graph. AtomOra stores extracted insights, connections, and questions—not raw transcripts. Ninety percent of conversation is noise. What matters is: which papers were discussed, what claims were made, what connections were identified, and what questions remain open. Over time this enables AtomOra to propose research directions grounded in your actual accumulated thinking, not generic literature review.

3. Architecture

AtomOra is a standalone Python application running as a macOS menubar daemon. It calls the Anthropic Messages API for deep conversation and runs a suite of local models for perception, gating, and voice. The architecture follows a three-module pipeline: Perception (always running, zero API cost), Gate (decides relevance), and Response (generates conversation).

3.1 Perception Module

The perception module runs continuously and never calls an external API. All processing is local.

- **Always-on microphone:** whisper.cpp with Whisper Large V3 Turbo on Apple Neural Engine. Continuous streaming transcription. silero-vad for voice activity detection.
- **Window/file monitor:** pyobjc NSWorkspace notifications detect the active PDF. pymupdf extracts full text locally.
- **Event-driven screenshot:** macOS screencapture triggered only by voice reference to visual content, highlight detection, or manual hotkey.
- **Pixel-diff change detection:** Local frame comparison detects page changes without API calls.

3.2 Gate Module

The gate decides whether an utterance is directed at AtomOra or is ambient noise. It combines three signals:

- **Semantic relevance (MiniLM-L6-v2):** Cosine similarity between the utterance embedding and the current page text. High similarity suggests the user is talking about the paper.
- **Address signal detection:** Keywords and patterns indicating the user is talking TO someone: question intonation, phrases like “你觉得呢”, “I think”, “what if”.
- **Attention state model:** Three states based on speech frequency and silence patterns. Deep focus: stay silent. Casual reading: brief observations. Active discussion: full conversational mode.

3.3 Response Module

Deep conversation goes through Claude (Sonnet) via the Anthropic Messages API with a colleague-persona system prompt. Voice output through local TTS (Kokoro 82M for English, Qwen3-TTS 0.6B for Chinese). Optional notification bubble for text fallback.

3.4 Local Vision (Qwen-VL)

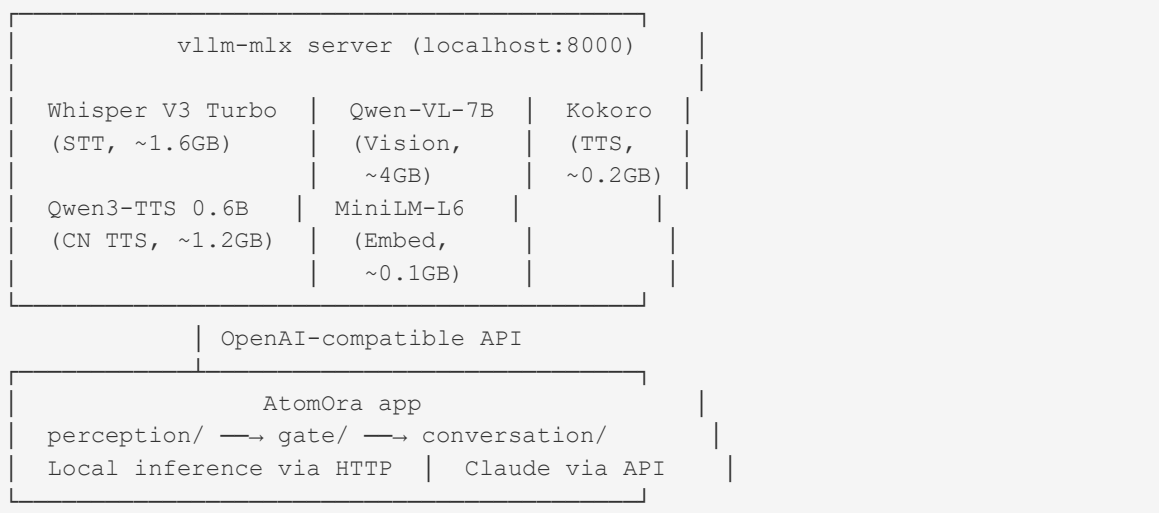
A local vision-language model (Qwen2.5-VL-3B or 7B, 4-bit quantized via MLX) handles visual understanding tasks that would otherwise require expensive Vision API calls. This includes: describing figures and charts in papers, determining what is currently visible on screen for gate decisions, and providing visual context to Claude when a full API call is warranted. The local VLM acts as a cheap triage layer—most visual questions can be answered locally, and only complex analytical questions escalate to Claude.

4. Local Inference Stack

All local models are served through a unified inference backend, eliminating the need for each module to load models independently. The target hardware is an M1 Max with 64GB unified memory.

4.1 Unified Server: vllm-mlx

vllm-mlx provides a single process that manages all local models with OpenAI-compatible API endpoints. It integrates mlx-lm (text LLMs), mlx-vlm (vision-language models), mlx-audio (STT and TTS), and mlx-embeddings (semantic search). The AtomOra application communicates with this server via HTTP on localhost, making all local inference calls uniform and all models hot-swappable.



4.2 Memory Budget

Model	Quant	Memory	Purpose
Whisper Large V3 Turbo	fp16	~1.6 GB	Speech-to-text, always-on streaming
Qwen2.5-VL-7B	4-bit	~4 GB	Visual understanding, figure description, gate
Qwen3-TTS 0.6B	bf16	~1.2 GB	Chinese text-to-speech with emotion control

Kokoro 82M	bf16	~0.2 GB	English text-to-speech, ultra-fast generation
MiniLM-L6-v2	fp32	~0.1 GB	Semantic embeddings for gate + briefing filter
Total		~7.1 GB	11% of 64 GB unified memory

The remaining 57 GB supports Adobe Acrobat, Chrome, VS Code, Slack, and any other applications running concurrently. All models are resident in memory with zero cold-start latency.

4.3 Chinese-English Bilingual Handling

The user naturally code-switches between Chinese and English during paper reading—reading English papers while discussing in Chinese, or mixing English terminology into Chinese commentary. This has specific implications for both STT and TTS.

For STT, Whisper Large V3 Turbo handles code-switching natively as a multilingual model. A decode-time prompt is injected with frequently used technical terms (hBN, SPE, cathodoluminescence, STEM, etc.) to reduce recognition errors on domain-specific vocabulary.

For TTS, language detection on Claude's response determines voice routing: primarily Chinese responses use Qwen3-TTS (superior Chinese prosody from Alibaba's training data), primarily English responses use Kokoro (fast, natural English voices). Both models are loaded simultaneously at negligible combined memory cost.

4.4 Three-Tier Cost Model

Most operations are free. API calls occur only on specific triggers.

- **Tier 0 (free, continuous):** File monitoring, window detection, PDF text extraction, pixel-diff, local Whisper STT, local TTS, local VLM gate, local embeddings. Cost: $0.
- **Tier 1 (cheap, on user action):** Text events to Claude—page context, highlighted text, conversation turns. Tens to hundreds of tokens per event. Cost: fractions of a cent.
- **Tier 2 (expensive, explicit trigger only):** Screenshots sent to Claude Vision API when local VLM cannot resolve a visual question. 1000–1500 tokens per image. Only triggered when user explicitly references a visual element that requires deep analysis.

Estimated cost for a 1-hour reading session: $0.01–$0.05 with Sonnet. Sustainable for daily use. With the $200/month Max 20x subscription, all Claude API calls for both the reading companion and academic briefing system are covered within the existing plan allocation.

5. Phase 1: Talking Sidebar (MVP)

Build time: one weekend (~12 hours). The minimum viable product that proves the core interaction model works.

Deliverable: A menubar icon. Open a PDF in Adobe Acrobat, press a hotkey once to load the paper context. Then just talk. AtomOra talks back. Phase 1 still uses a hotkey trigger (always-on ambient listening comes in Phase 2).

Milestone breakdown

- **Menubar app skeleton (rumps):** 2 hours. Status icon, basic event loop, global hotkey registration (Cmd+Shift+R).
- **PDF loader:** 2 hours. Detect frontmost PDF via pyobjc, extract full text with pymupdf, inject into conversation context.
- **Voice input:** 3 hours. whisper.cpp + silero-vad. Hotkey triggers recording, silence detection ends recording, transcription returned.
- **Claude integration:** 2 hours. Anthropic Messages API with colleague system prompt. Paper text as context, transcription as user message, response streamed back.
- **Voice output:** 1 hour. Day 1: macOS say (zero-setup). Day 2–3: upgrade to Kokoro via mlx-audio.
- **Conversation loop:** 2 hours. Wire all components together. Maintain last 20 exchanges as conversation history.

6. Phase 1.5: Academic Briefing System

Build time: half a day to one day. An asynchronous, standalone module that runs independently as a daily cron job. It monitors new publications, filters for personal relevance, generates contextualized analysis, and delivers results to a Slack channel or mobile push notification. This module can be built and deployed before any of the real-time companion features, and it starts delivering value immediately.

6.1 Three-Stage Pipeline

Stage 1 — Harvest: A daily cron job (default 05:00 EST) queries arXiv (free, no key), Springer Nature (free API key via dev.springernature.com), CrossRef (free, no key), and OpenAlex (free, no key) for papers published in the previous 24 hours. Only metadata and abstracts are collected. No institutional credentials required.

Stage 2 — Filter: Each paper's abstract is embedded with MiniLM-L6-v2 and scored against a research profile centroid. The profile is constructed from 10–20 manually curated anchor papers (v1) or auto-generated from the knowledge graph (v2). Scoring combines semantic similarity (0.5), keyword match (0.3), and journal authority tier (0.2). Adaptive threshold targets 3–8 papers per day. Runs entirely locally at zero cost.

Stage 3 — Contextualize: For each paper passing the filter, Claude (Sonnet) receives the abstract plus research profile context and answers three questions: relevance to the user's specific work, notable methodology, and tension or convergence with existing positions. Daily cost: approximately $0.04 (5–8 papers, ~2000 tokens each).

6.2 Delivery

Primary delivery is a Slack channel (#atomora) via incoming webhook. The full daily briefing is posted as a single message with rich formatting: journal name, title, authors, contextualized analysis, relevance stars (1–5), and a recommendation (skip / skim / read methods / read fully / read + discuss). Secondary delivery via Pushover ($5 one-time app) or ntfy (free) for mobile push notifications.

An optional high-priority interrupt delivers papers scoring above 0.90 relevance immediately rather than waiting for the morning digest, ensuring time-sensitive developments (e.g., a competitor publishing closely overlapping results) reach you promptly.

6.3 Evolution

v1 uses a static research profile from manually curated anchor papers. When Phase 4's knowledge graph comes online, the profile auto-generates from accumulated reading sessions. v3 adds a feedback loop: the paper-reading companion (Phases 1–3) detects which briefed papers are actually read, automatically retraining the relevance model based on reading behavior rather than explicit feedback.

7. Phase 2: Ambient Listening

Build time: ~12 hours across one week. Removes the hotkey requirement. AtomOra listens continuously and decides autonomously when to respond.

- **Continuous STT:** 3 hours. Always-on streaming Whisper via whisper.cpp with Neural Engine acceleration.
- **Relevance filter:** 3 hours. MiniLM embeddings compare each utterance to current page text via cosine similarity.
- **Address detection:** 2 hours. Rule-based keyword matcher for conversational signals.
- **Auto PDF detection:** 2 hours. Watch NSWorkspace window changes, auto-load new papers.
- **Attention state:** 2 hours. Speech frequency heuristic determines deep focus / casual reading / active discussion mode.

Deliverable: No buttons. Mutter "这个sample size太小了" and AtomOra responds. Take a phone call and AtomOra stays quiet.

8. Phase 3: Visual Awareness

Build time: ~11 hours across weeks 3–4. AtomOra gains the ability to see what you see.

- **Smart screenshot:** 3 hours. Pixel-diff detects page changes; screenshot captured only on change events.
- **Highlight detection:** 3 hours. Monitor clipboard/selection changes to detect when text is highlighted in the PDF.
- **Vision integration:** 2 hours. Local Qwen-VL describes figures; complex visual questions escalate to Claude Vision API.
- **Proactive comments:** 3 hours. AtomOra occasionally comments unprompted based on what it observes on screen and in the conversation history.

Deliverable: Reference a figure and AtomOra sees it and comments. Highlight a passage and AtomOra notices and responds.

9. Phase 4: Memory & Knowledge Graph

Build time: ~15 hours across month 2+. This is the phase that transforms AtomOra from a session-based tool into a longitudinal research intelligence.

9.1 Post-Session Extraction

After each reading session, Claude analyzes the conversation asynchronously (batch processing overnight) and extracts structured knowledge: papers discussed (with file paths and DOIs), key claims and observations, connections to other work, methodological critiques, open questions and research gaps, and the user's preferences and judgments.

9.2 Storage Architecture

NetworkX graph with nodes for papers, concepts, methods, and researchers. Edges encode relationships extracted from discussions (cites, contradicts, extends, uses-method, etc.). ChromaDB vector store enables semantic search over session summaries and extracted insights. The graph is queryable both structurally (what papers discuss vacancy engineering in hBN?) and semantically (what have I said about characterization standards?).

9.3 Context Injection

Before each new reading session, AtomOra queries the knowledge graph for papers and concepts related to the current document. Relevant context is injected into the system prompt, enabling responses like: "This group's previous work used a different characterization protocol—you flagged issues with it two months ago. Want me to pull up that discussion?"

9.4 Long-Term Vision: Academic Footprint Map

After months of accumulation, the knowledge graph becomes an "academic footprint map"—like GPS traces showing where your body has been, it shows where your mind has been. AtomOra knows your research taste, methodological preferences, open questions, and the trajectory of your thinking over time. It can extrapolate forward: suggest papers to read, identify literature gaps aligned with your expertise, and draft research proposals grounded in your actual thinking trajectory. The transition is from reading companion to research intelligence platform.

10. Automated Development

AtomOra's development leverages two automation strategies: Ralph Wiggum loops for mechanical code generation and a synthetic evaluation pipeline for quality assurance. Both run on the existing $200/month Claude Max 20x subscription with no additional API cost.

10.1 Ralph Wiggum Loop

Ralph is an autonomous development loop where Claude Code iterates on a task until programmatically verified completion criteria are met. Instead of manually prompting Claude Code step by step, you define a PRD (Product Requirements Document) with user stories, acceptance criteria, and dependencies, then let the loop run overnight while you sleep.

Each iteration spawns a fresh Claude Code instance that sees the modified files and git history from previous runs. Memory persists through a shared markdown progress file (TASKS.md)

where Claude records what was done and what should be done next. The loop continues until all acceptance criteria pass or the maximum iteration count is reached.

When to use Ralph

- **Mechanical scaffolding:** API client wrappers, file structure creation, boilerplate code, error handling, retry logic.
- **Test coverage:** Writing unit tests, integration tests, achieving coverage targets.
- **Batch operations:** Migrating code patterns, standardizing interfaces, implementing data models.
- **Briefing pipeline plumbing:** Implementing harvesters for each data source, filter scoring, Slack formatter.

When NOT to use Ralph

- **Aesthetic decisions:** System prompt persona tuning, notification styling, voice selection, gate sensitivity thresholds.
- **Environment configuration:** pyobjc setup, whisper.cpp compilation on M1 Max, rumps integration with macOS.
- **Interaction design:** Anything requiring human judgment about how AtomOra should feel to use.

Workflow

Build the skeleton manually (environment config, aesthetic decisions, core interaction loop). Then hand off mechanical implementation to Ralph overnight. Review PRs in the morning. Use interactive Claude Code during the day for persona tuning and interaction refinement.

```
# Example: Ralph loop for Phase 1.5 briefing pipeline
# Install Ralph plugin
/plugin marketplace add anthropics/claude-code
/plugin install ralph-wiggum@claude-plugins-official

# Define the loop
/ralph-loop "Implement the arxiv harvester in
  briefing/sources/arxiv.py per the spec in CLAUDE.md.
  Write tests in tests/test_arxiv.py. Output
  <promise>DONE</promise> when pytest passes."
  --max-iterations 25
  --completion-promise "DONE"
```

10.2 Synthetic Evaluation Pipeline

Before deploying any changes to AtomOra's persona, knowledge graph extraction logic, or context injection strategy, an automated evaluation pipeline validates quality. This is CI/CD for AI personality.

Architecture

The pipeline uses three distinct Claude instances (or a mix of Claude and Gemini for cross-model objectivity):

- **Questioner agents:** Multiple Claude instances with diverse cognitive styles act as simulated users. Each has a distinct personality: one is methodical and detail-oriented, asking deep follow-up questions; one is creative and lateral, making unexpected cross-domain connections; one is skeptical, pushing back on claims and demanding evidence. This stress-tests AtomOra's ability to handle varied interaction styles.
- **AtomOra (system under test):** The research companion with its current system prompt, knowledge graph, and context injection running in the loop.
- **Judge agent:** A separate Claude or Gemini instance that evaluates the quality of AtomOra's responses on dimensions like: relevance to the paper, accuracy of technical claims, appropriate use of knowledge graph context, persona consistency (colleague vs. assistant), and actionability of insights.

Test Corpus

The user's Zotero library serves as the ground truth corpus. Papers already collected over years of research represent the actual research trajectory. The pipeline imports metadata and annotations from Zotero, generates simulated reading sessions across different papers, and evaluates whether AtomOra produces meaningfully different responses as the knowledge graph accumulates—testing that memory actually improves interaction quality over time.

Execution

The entire pipeline is itself a Ralph loop task. Define completion as: all test scenarios executed, evaluation scores logged, regression report generated. Run overnight. Morning review focuses on flagged regressions and edge cases, not manual testing.

```
# Eval pipeline structure
atomora/
  eval/
    personas/
      methodical.yaml     # Deep follow-up questioner
      lateral.yaml        # Cross-domain connector
      skeptical.yaml      # Evidence-demanding critic
    scenarios/
      single_paper.py     # One paper reading session
      multi_session.py    # Knowledge accumulation test
      briefing_quality.py # Academic briefing accuracy
    judge.py              # Evaluation scoring (Claude/Gemini)
    runner.py             # Orchestrates full eval suite
    report.py             # Generates regression report
```

11. Project Structure

```
atomora/
├── main.py                    # Entry point, menubar app
├── perception/
│   ├── microphone.py          # Always-on STT (whisper.cpp)
│   ├── window_monitor.py      # Active window/file detection
│   ├── pdf_extractor.py       # Text extraction (pymupdf)
│   └── screenshot.py          # Event-driven screen capture
├── gate/
```

```
│   ├── relevance.py           # Semantic similarity filter
│   ├── address_detect.py      # Is user talking to me?
│   └── attention.py           # Focus state estimation
├── conversation/
│   ├── claude_client.py       # Anthropic API wrapper
│   ├── prompts.py             # System prompts (colleague persona)
│   └── context.py             # Context window management
├── voice/
│   ├── tts.py                 # TTS output (Kokoro + Qwen3-TTS)
│   └── notification.py        # macOS notification bubbles
├── briefing/
│   ├── harvester.py           # Stage 1: fetch from APIs
│   ├── sources/
│   │   ├── arxiv.py           # arXiv Atom feed client
│   │   ├── springer.py        # Springer Nature (sprynger)
│   │   ├── crossref.py        # CrossRef metadata API
│   │   └── openalex.py        # OpenAlex REST API
│   ├── filter.py              # Stage 2: score + rank
│   ├── contextualizer.py      # Stage 3: Claude analysis
│   ├── delivery/
│   │   ├── slack.py           # Slack webhook formatter
│   │   └── pushover.py        # Mobile push (optional)
│   ├── profile.py             # Research profile management
│   └── scheduler.py           # Cron job wrapper
├── memory/
│   ├── extractor.py           # Post-session knowledge extraction
│   ├── graph.py               # NetworkX knowledge graph
│   └── vectorstore.py         # ChromaDB semantic search
├── eval/
│   ├── personas/              # Questioner personality configs
│   ├── scenarios/             # Test scenario scripts
│   ├── judge.py               # Evaluation scoring
│   ├── runner.py              # Orchestrator
│   └── report.py              # Regression report generator
├── config/
│   ├── sources.yaml           # Harvest query config
│   ├── profile.yaml           # Anchor papers + keywords
│   └── secrets.yaml           # API keys (gitignored)
├── data/
│   ├── embeddings/            # Cached abstract embeddings
│   ├── history.json           # Previously briefed papers
│   └── graph/                 # Knowledge graph data
├── CLAUDE.md                  # Instructions for Claude Code
└── requirements.txt           # Python dependencies
```

12. Tech Stack

Component	Technology	Rationale

Language	Python 3.11+	Best ecosystem for ML, audio, macOS integration
Desktop integration	pyobjc + rumps	rumps for menubar; pyobjc for macOS APIs
STT	whisper.cpp (Large V3 Turbo)	Neural Engine optimized; 6x faster than full model
VAD	silero-vad	Lightweight, accurate, no GPU
TTS (English)	Kokoro 82M (mlx-audio)	Sub-second generation, 24 voice presets
TTS (Chinese)	Qwen3-TTS 0.6B (mlx-audio)	Superior Chinese prosody, emotion control, voice cloning
Local VLM	Qwen2.5-VL-7B (MLX 4-bit)	Visual gate, figure description, screen understanding
PDF extraction	pymupdf (fitz)	Fast, reliable, pure Python
LLM	Claude Sonnet (Anthropic API)	Deep conversation, contextualization, knowledge extraction
Embeddings	MiniLM-L6-v2 (local)	Semantic similarity for gate + briefing filter
Knowledge graph	NetworkX + ChromaDB	Graph structure + vector search
Local inference server	vllm-mlx	Unified OpenAI-compatible API for all local models
Screenshot	macOS screencapture	Built-in, silent, fast
Academic APIs	sprynger, feedparser, habanero, pyalex	Springer Nature, arXiv, CrossRef, OpenAlex
Delivery	Slack webhook + Pushover	Zero-infrastructure notification delivery
Development automation	Ralph Wiggum (CC plugin)	Autonomous overnight code generation loops

13. Build Sequence

The build follows a principle of progressive value delivery. Each phase produces something usable before the next begins. The sequence is ordered by impact-to-effort ratio, not by technical dependency.

Phase	Deliverable	Time	Method
1	Talking sidebar: hotkey, voice in/out, Claude conversation with paper context	Weekend	Manual (env config, aesthetic choices)
1.5	Academic briefing: daily Slack digest of relevant new papers	Half day	Manual skeleton + Ralph for API plumbing

2	Ambient listening: no buttons, semantic gate, auto PDF detect	Week 2	Manual (gate tuning requires feel)
3	Visual awareness: screenshot, highlight detection, figure understanding	Weeks 3–4	Ralph for plumbing, manual for VLM integration
4	Knowledge graph: post-session extraction, context injection, longitudinal memory	Month 2+	Ralph for data pipeline, manual for extraction prompts
Eval	Synthetic eval pipeline: multi-persona stress testing with Zotero corpus	Parallel	Ralph loop (entire pipeline is automatable)

14. Day 1 Checklist

```
mkdir atomora && cd atomora
python3 -m venv venv && source venv/activate
pip install anthropic pymupdf silero-vad rumps pyobjc
brew install whisper-cpp
pip install mlx-audio  # Kokoro TTS
# Create CLAUDE.md with project spec
claude  # Start Claude Code
```

First prompt to Claude Code:

```
> Read CLAUDE.md. Let's build Phase 1. Start with main.py:
  a menubar app using rumps that shows a status icon.
  Add a global hotkey (Cmd+Shift+R) that triggers
  "start listening" mode. When triggered, record audio
  until 1.5s of silence, transcribe with whisper.cpp,
  and print the transcription. That's the first milestone.
```

Then iterate: add PDF detection, add Claude API integration, add voice output, add conversation history, test with a real paper, tune the system prompt until it feels like a colleague.

AtomOra is not a tool you use. It is a colleague you work with. The difference is that a tool waits for instructions; a colleague shares your context, has opinions, and grows with you over time. Everything in this specification—the ambient listening, the knowledge graph, the academic briefing, the evaluation pipeline—serves that single idea.