COMPANION INTELLIGENCE

Companion Memory: Architecture of a Personal Memory Appliance

A technical overview

July 2026 · ci.computer

© 2026 Companion Intelligence · ci.computer

EXECUTIVE SUMMARY

A memory that belongs to you

Companion Memory is a personal memory appliance. It is a database, a search engine, a knowledge graph, and a prediction engine, and it runs on hardware you own — a Companion Core, a NAS, or a workstation. It captures what you do across your browser, your desktop, and your exported service archives, stores all of it in a single local database, and makes it useful: to you, through search and exploration, and to your AI agents, through a standard protocol.

The architecture rests on one separation. A cloud plane handles identity, entitlements, and the app catalog. A personal data plane — your appliance — holds the data. Your data lives on your appliance; the cloud plane sees identity, entitlements, and catalog metadata only.

Everything computationally interesting happens on the appliance. Embeddings are computed locally. The language models that summarize, label, and answer questions run locally. The prediction engine that learns your daily rhythm is a model of one: trained on your history, fine-tuned every night on your own CPU, and never shared with anyone else.

This paper walks through the system end to end: the two-plane model, the capture pipeline, storage and the knowledge graph, hybrid retrieval, the agent interface, and the prediction engine. It closes with the things we deliberately do not do.

84+	250+	30+	128 GB / 2 TB
services to import from	one-click apps	local models	Companion Core memory / storage

SECTION 1

The two-plane model

Most personal-data products put the data where the company is. We put the data where you are, and we drew the line in the architecture, not in a policy document.

The **cloud control plane** is CI Portal. It is the identity provider you sign in with, the registry that knows which apps your organization is entitled to, and the distribution edge for the app catalog. It also routes inbound traffic to your appliance through a secure tunnel, so the appliance never has to be directly exposed on the public internet. The control plane does not store user content.

The **personal data plane** is your appliance. It runs Companion Memory — the database, search, knowledge graph, and local language models — alongside an app runtime that installs and supervises apps from a catalog of 250+ one-click apps, and the prediction engine described in Section 6.

The framing we hold ourselves to is simple: **your data lives on your appliance; the cloud plane sees identity, entitlements, and catalog metadata only.** Signing in tells the cloud who you are. Installing an app tells it what you are entitled to. Neither tells it what you did today.

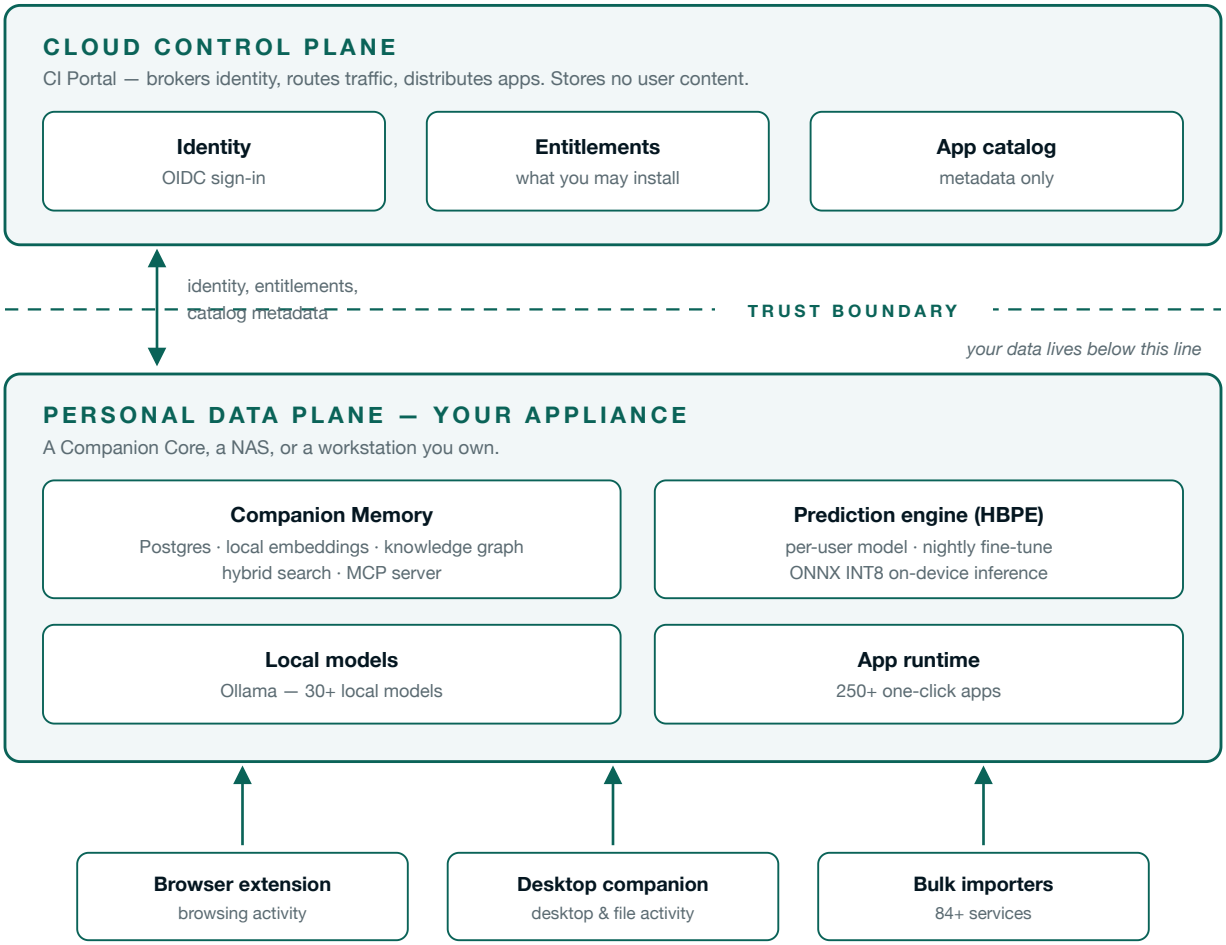

Figure 1 — The two-plane model. Identity and app distribution live in the cloud; your data lives on the appliance.

SECTION 2

Capture: one event stream

A memory is only as good as what reaches it. Companion Memory captures your digital life through three surfaces, all of which converge on a single, unified event stream on the appliance.

The browser extension

The browser is where most digital life happens. Our extension captures your browsing activity — the pages you visit and the things you save — and writes it directly to your appliance. It authenticates with the same identity you sign in with; there is no separate account and no intermediary service.

The desktop companion

A lightweight tray application observes your working day on the desktop: application activity, file events, and screen context. It queues locally and delivers to the appliance, so a dropped connection loses nothing.

Bulk importers

Years of your history already exist — inside the services you have used. Our importers read the archives those services let you export, covering 84+ services. The canonical first run is Google Takeout: export your archive, point the importer at it, and years of mail, calendar, location, and activity history land on your own timeline, in your own database. Import is idempotent — re-running an import recognizes what is already there instead of duplicating it.

The unified event stream

Every capture surface writes to the same contract: a batched events endpoint on the appliance. Each event carries its source system, a source-scoped identifier, a timestamp, and a typed payload. The pipeline is idempotent on the source identity, so every writer — extension, desktop, importer — can safely retry. An append-only capture log preserves the raw record before anything downstream interprets it.

From that stream, everything else follows automatically: events land in the database, embeddings are computed, entities are extracted and linked, and the search indexes update. Capture is the only step you have to set up. The rest is the appliance's job.

SECTION 3

Storage and the knowledge graph

Companion Memory keeps your life in **one local PostgreSQL database** on the appliance. Vector search (pgvector) and time-series storage (TimescaleDB) run as extensions inside that same database, so your semantic index and your activity timeline live next to the data they describe — one store, one backup surface, one owner: you. Files and media sit alongside it in an S3-compatible object store, also on the appliance.

Embeddings, computed locally

Semantic search requires embeddings, and embeddings are usually where personal data quietly leaks to a third-party API. Ours never leave the box. An Ollama runtime on the appliance computes 768-dimensional embeddings with `nomic-embed-text`, feeding a single unified embedding store that covers every entity type — messages, events, people, places, files, notes. The same runtime hosts the chat and summarization models; the appliance supports 30+ local models.

The entity graph

Raw events are a diary. A memory needs a cast. Companion Memory continuously distills captured data into an entity graph: **people, places, and things**, connected by typed relationships and resolved across sources through stored identifiers.

- **People** emerge from your communications and your calendar: the senders you correspond with and the attendees on your events become person records, linked to everything they appear in.
- **Places** emerge from events and location history, so "where was I" and "what happened here" are graph queries, not archaeology.
- **Things** — concepts, topics, media, items — connect the rest, giving agents stable handles for the recurring furniture of your life.

Agent conversations feed the same graph. Memory fragments written by your agents are periodically compacted by a local model into structured facts and entities, so what your companion learns in conversation becomes part of the same graph it queries later.

SECTION 4

Retrieval: finding what you remember

You remember your life in fragments: a word from a page title, the gist of a conversation, roughly when, roughly where. Retrieval has to meet all of those, so Companion Memory searches three ways at once.

- **Keyword** — weighted full-text search for when you know the words. Titles weigh more than body text; exact terms win.
- **Semantic** — vector similarity over the local embeddings for when you only know the meaning. "That article about batteries degrading in the cold" finds the page whose title never mentioned either word.
- **Boosted** — keyword relevance amplified by semantic similarity. Precision where you have words, recall where you have meaning.

An automatic mode routes each query: short, specific queries go to keyword; long natural-language questions go semantic; everything in between gets the boosted blend. You search; the system picks the mechanics.

The memory explorer

Search results are one lens. The memory explorer in the web UI gives you four more, all views over the same store:

- **Timeline** — your history in order: the when.
- **Map** — memories placed geographically, filterable by region: the where.
- **Graph** — the entity graph made visible: people, places, and things and the connections between them.
- **Gallery** — your visual record, browsable.

Searches can be saved and returned to. Who, what, when, where — answered from your own record, on your own hardware.

Recall for agents

When an agent asks for context, retrieval goes further than a ranked list. A recall pass seeds a boosted search, resolves the entities it finds, expands one hop through the graph to pull in connected people, places, and facts, and folds in recent conversation memory. The result is assembled context — not ten blue links — delivered to the agent when it asks.

SECTION 5

Agents: acting with context

A memory that only you can read is a scrapbook. Companion Memory is built to be read by your AI agents — on your terms, over a standard protocol, on your hardware.

An MCP server on the appliance

Companion Memory ships with a built-in **Model Context Protocol (MCP)** server using the Streamable HTTP transport. MCP is an open standard, so any MCP-capable agent or client can connect with a key issued on your appliance and work with your memory — no proprietary SDK required.

Twelve native intents, six domains

Rather than exposing raw database plumbing, the appliance publishes a curated capability layer: **12 native intents across 6 domains** — memory, knowledge, people, places, calendar, and media. Each intent has a dotted name — `memory.recall`, `people.find`, `calendar.findEvents` — a description, a typed parameter schema, and the entity types it consumes and produces. The full catalog is machine-readable: agents discover what the appliance can do by fetching it, not by hard-coding it.

The domains map to how you actually use a memory: recall and store facts (memory), search the whole corpus (knowledge), find people and relationships (people), find locations you have been (places), reason over your timeline (calendar), and find photos, videos, and media (media).

Reference harnesses: Hermes and OpenClaw

We maintain two reference agent harnesses that show the integration end to end: **Hermes** (Python) and **OpenClaw** (JavaScript). Both connect a local agent loop to Companion Memory — reading and writing agent memory, discovering the intent catalog, and receiving assembled context from the recall pipeline. They run against the same local models as everything else on the appliance.

Forecast context injection

Agents get one more thing the rest of the industry cannot offer: your likely day. When an agent requests memory context, the appliance includes a *Predicted Rest Of Day* block from the prediction engine (Section 6). An agent planning your afternoon sees where the afternoon is probably headed. All of this is pull-based: context is assembled when the agent asks for it, and forecasts are served on request — through the UI, the API, and agent context.

SECTION 6

The prediction engine (HBPE)

Memory tells you what happened. The prediction engine — HBPE, the **Hierarchical Behavioral Prediction Engine** — learns the shape of your days and forecasts what comes next. It runs as a dedicated service inside the appliance, and it follows one rule without exception: **one user, one model.** Your model is trained on your history, tuned on your hardware, and shared with no one.

From raw events to an activity hierarchy

A small language model running on the appliance labels each captured item — calendar events, browser activity, hourly activity summaries — into a behavioral hierarchy. The top level is fixed: **8 classes** (work, exercise, social, errands, leisure, transit, sleep, and unknown). Below that sits a living, per-user vocabulary of discovered clusters — "deep focus," "team standup," "gym" — created the first time they appear in your data and retired automatically when they go unused. The hierarchy is yours: two users' models share the 8 classes and nothing else.

Days as sequences

Each day is serialized into **48 half-hour buckets**, framed by start-of-day and end-of-day markers. Calendar events carry full confidence; buckets covered only by hourly summaries carry reduced confidence, so quiet stretches still contribute signal without outvoting hard facts.

A transformer that is small on purpose

The forecasting model is a transformer with a model dimension of 64 and 2 layers. That is not a compromise — it is the design. The training corpus is one person's days, not the internet; a large model would memorize your history rather than learn its rhythm. Small is also what makes local ownership practical: the model trains and fine-tunes on the appliance's CPU, quantizes cleanly, and leaves the rest of the machine free to do its day job.

Nightly fine-tuning, on your CPU

Every night, a durable workflow labels anything new, prunes stale vocabulary, and fine-tunes your model with a per-user LoRA adapter on the day you just lived — a step **designed to finish in under five minutes on CPU**. No GPU farm, no training queue in someone else's data center. The tuned model is exported to ONNX and quantized to INT8, and that compact artifact serves all inference on the appliance.

The rest-of-day forecast

The output is a forecast of your next 24 hours at 30-minute resolution: for each half-hour slot, a probability distribution over the behavioral classes and your personal sub-activities, aggregated over recent history with recency weighting and a same-weekday emphasis. It renders as a strip in the web UI and flows into agent context on request.

> **Auditable by design.** Every forecast the engine issues is logged on the appliance, tagged with the model version that produced it, so it can be checked against what actually happened. A prediction engine you own should be one you can audit.

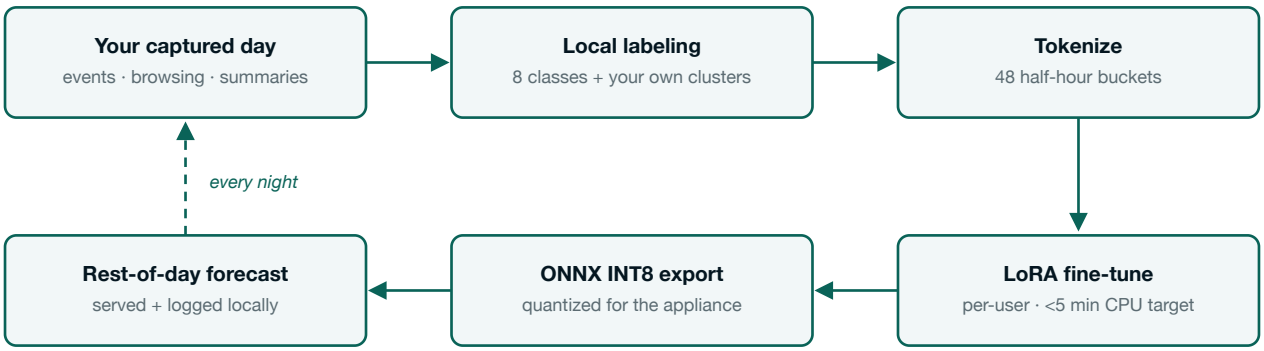

Figure 2 — The nightly loop. Your day becomes training data; your model becomes a little more yours.

PROPERTY	DESIGN
Top-level activity classes	8 fixed classes + discovered per-user clusters
Day resolution	48 buckets of 30 minutes
Model size	Transformer, model dimension 64, 2 layers — small on purpose
Personalization	Nightly per-user LoRA fine-tune; designed to finish in under 5 minutes on CPU
Inference	ONNX INT8, on the appliance
Accountability	Every forecast logged on the appliance with its model version

The recurrence layer: week ahead and due signals ROLLING OUT

Beyond 24 hours, the right question changes. Nobody needs to know whether the gym is at 2:00 or 2:30 next Thursday; they need to know it is probably Thursday. The recurrence layer answers longer-horizon questions with **transparent statistics, not a neural model**.

Each night, the appliance rebuilds a statistical profile for every recurring habit in your vocabulary from up to roughly 180 days of history: which weekdays it lands on, the gaps between occurrences, and a confidence score for whether it is genuinely periodic at all. From those profiles come two signals:

- **Week ahead** — for each day of the coming week, the probability that a habit occurs, from its renewal statistics blended with its day-of-week base rate. Gym: likely Tuesday.
- **Due and overdue** — for habits with real periodic structure, an estimate of when they are due again from the typical gap, and an overdue flag once the gap stretches past that habit's usual upper range. You usually see friends weekly; it has been 16 days.

Because the layer is arithmetic over your own history, every number it produces can be traced back to the occurrences that produced it. These signals surface through the same pull-based interfaces as the daily forecast.

SECTION 7

What we deliberately don't do

An architecture is defined as much by what it refuses. Three refusals define this one.

No cloud inference on personal data

Labeling, embedding, summarization, search, chat, and prediction all run on the appliance. When your data is being computed on, the computer is yours. The cloud plane brokers identity, checks entitlements, and distributes apps; it does not store user content, and it is not in the inference path for your memory.

No shared or pooled models

There is no communal model quietly averaging its users. The prediction engine is a model of one: your activity vocabulary is discovered from your data, and your nightly fine-tune sees your token stream and nobody else's. The base models the appliance runs — for chat, embeddings, and labeling — are open models hosted locally, not shared services.

No training on customers' data for other customers

Your history tunes your model. It is never used to train, tune, or improve a model that serves anyone else. Your captured life is not a corpus, and it is not the product.

> These are architectural properties, not policy promises. The database is on your disk. The models are on your box. The fine-tune runs on your CPU. Changing any of that would require changing the product, not a paragraph in a terms-of-service document.

Companion Memory is our answer to a simple question: what would personal AI look like if the person owned it? It looks like a database you can open, a model trained only on you, and a forecast you can check against your own day. It looks like an appliance.

Learn more: ci.computer/memory · docs.ci.computer · support@ci.computer