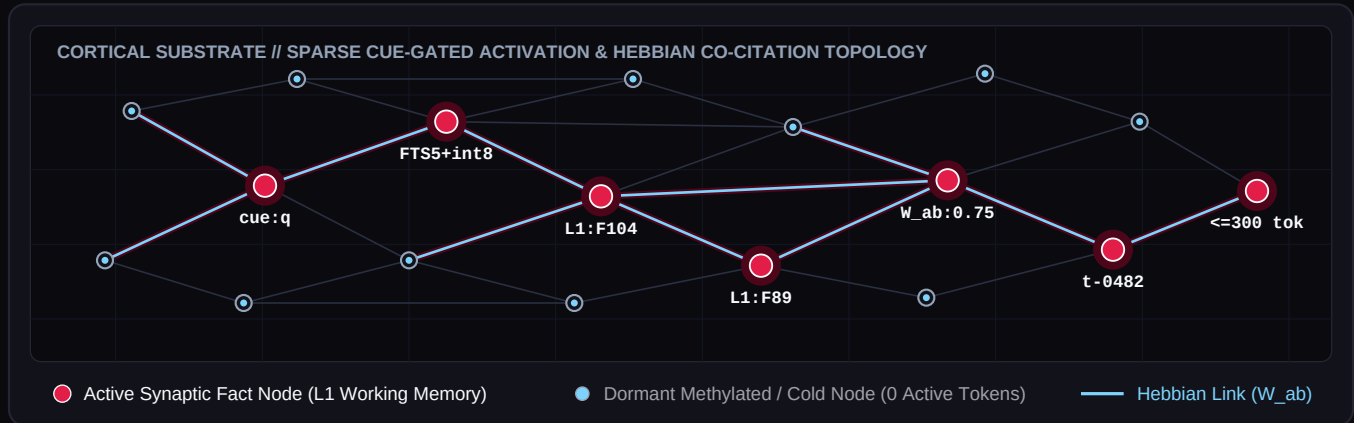


XBrain

Native Personal Memory Substrate for Grok Bot

Field Notes & Technical Architecture Concept for the xAI Team

VERSION 1.1 — COMMUNITY POWER-USER CONTRIBUTION // OCTOBER 2026



00 // A Note of Appreciation & Context for the xAI Team

I am sharing this write-up simply as an enthusiastic daily user of Grok, X, and the Grok Bot app (**on a fixed weekly usage pool**) who wants to help xAI build the best personal AI platform in the world. There is no commercial ask or expectation here—everything below is offered freely as field notes and a working reference design in case any of these ideas are useful to the team.

Over the past several months, I have been running a multi-agent setup inside Grok Bot: a Chief-of-Staff bot coordinating specialist bots (Research Analyst, X Media, Operations, and a Brain Conductor) alongside scheduled daily routines on the shared cloud computer. While working iteratively with Grok and Gemini to build a local memory repository (the `brain/` directory) so my bots never drop context, we noticed a counterintuitive pattern: **our weekly usage pool was depleting rapidly even though our database queries were only returning 1,200 to 2,700 tokens per check.**

When we traced the root cause, we found that the usage burn came from how multi-agent workflows interact with the Grok Bot runtime harness:

1. **Persistent re-reading of tool outputs** that remain in each bot's chat history across turns
2. **Full LLM wake-ups** for scheduled routines where the underlying check is deterministic Python
3. **Inter-bot context duplication ("Fleet Tax")** when coordinating tasks across multiple bots

By designing a local-first memory and coordination pattern ("**XBrain**") on the shared cloud VM, we were able to solve these bottlenecks while creating a portable, **<100 MB** personal memory container that can grow over decades and eventually mount into embodied systems like **Tesla Optimus**. Below is the complete product and technical summary of how it works.

01 // Suggested Product Surface (Optional Native Primitives)

If xAI ever wanted to offer this capability natively so any Grok Bot user can benefit without custom setup, five lightweight primitives would cover the entire workflow:

1. **One-Click "Enable XBrain"**: Provisions a local `xbrain/` directory on the user's shared cloud computer and registers four built-in tools: `brain_check`, `brain_recall`, `brain_expand`, and `brain_ticket`.

2. **Ephemeral Tool Output (- -ephemeral):** Injects memory lookup results only for the current turn (T) and automatically strips them from the persisted chat transcript before the next turn (T+1). This preserves KV-cache prefix hits on prior turns and prevents CLI tool outputs from snowballing in chat history.
3. **Routine Pre-Check Hook (precheck_cmd):** Allows scheduled Routines to run a local Python or shell script on the cloud VM first (zero LLM tokens). The Grok LLM is woken **only if the script returns a non-zero exit code**.
4. **"Molt Thread (Keep Brain State)":** A one-click button (or optional 20-turn auto-rotation) that saves active task state to the brain and starts a fresh, zero-token chat thread without losing durable context.
5. **Universal AI History Importer:** Drag-and-drop ingestion for historical exports from **ChatGPT, Claude, Gemini, and Grok**. Raw transcripts are archived into cold storage (zero active tokens), while durable personal facts and preferences are distilled into the user's XBrain.

02 // Technical Architecture

2.1 File System Layout & Source-of-Truth Invariant

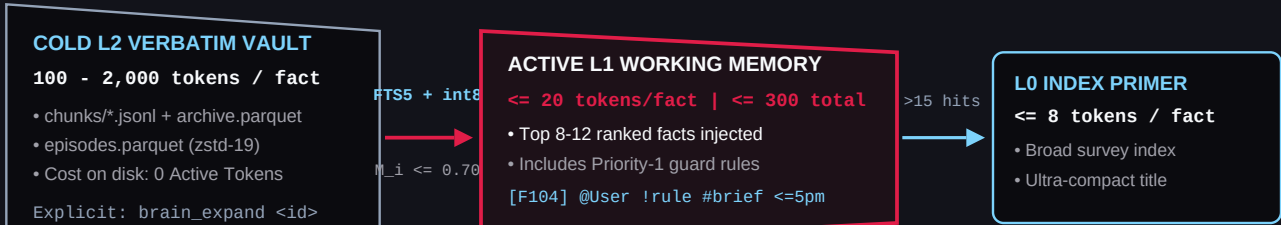
```
DIRECTORY LAYOUT // xbrain/

xbrain/
+-- facts/                # Append-only source of truth (human-auditable, delta-syncable)
|   +-- codons.json       # Versioned dictionary mapping recurring entities/verbs to short symbols
|   +-- chunks/*.jsonl    # Domain files storing L0, L1, L2 + methylation score
|   +-- relations.jsonl   # Append-only bi-temporal entity edges (valid_from, valid_to, provenance)
|   +-- links.jsonl       # Co-activation weights between memory chunks
+-- store/                # Derived, rebuildable indexes (regenerated via reindex)
|   +-- xbrain.db          # SQLite (FTS5 keyword index + int8 embeddings + Bi-Temporal Graph tables)
|   +-- archive.parquet   # Cold L2 facts compressed with zstd-19
|   +-- episodes.parquet  # Cold vault of raw multi-AI chat history (zstd-19, 0 active tokens)
+-- runtime/
|   +-- threads.jsonl     # Task tickets (t-xxxx) for pointer-only bot handoffs
|   +-- scratch/          # File-buffered pads (/scratch/t-xxxx.md) for heavy bot output
+-- telemetry/
    +-- public_health.json # Aggregate, non-PII metrics only
```

Core Invariant: The append-only JSONL files in `facts/` are the single source of truth. The SQLite database (`xbrain.db`) and Parquet archives in `store/` are purely derived indexes that can be rebuilt from scratch at any time via `brain.py reindex - -full`.

2.2 Cue-Gated, Multi-Resolution Memory (L0 / L1 / L2)

FIG 2.2 // CUE-GATED MULTI-RESOLUTION MEMORY FUNNEL (mRNA TRANSCRIPTION ANALOGY) **HARD CAP: <= 300 TOKENS / CHECK**



BIOLOGICAL PRINCIPLE: Ribosomes transcribe condensed mRNA (L1) on cue rather than loading the entire chromosome (L2) into active context.

Inspired by how biological cells transcribe only a tiny gene segment into working **mRNA** rather than pulling an entire chromosome into the ribosome, every fact is stored at three resolutions simultaneously:

FACT SCHEMA // chunks/ops_briefs.jsonl

```
{
  "id": "F104",
  "chunk": "ops_briefs",
  "p": 1,
  "l0": "Weekly brief deadline rule",
  "l1": "@User !rule #briefs: due_day <= Fri 17:00 -> notify; no_override",
  "text": "Full verbatim rule and historical context explaining why the weekly brief deadline w
as set on 2026-03-14...",
  "methylation": 0.10,
  "created_at": "2026-03-14T09:00:00Z"
}
```

Tier	Role	Token Budget	Runtime Behavior
L0	Primer / Index	<= 8 tokens	Emitted for broad surveys when >15 candidate facts match a query.
L1	Active Working Memory	<= 20 tokens	Default injection tier. <code>brain_check</code> ranks facts inside triggered chunks (via SQLite FTS5 + <code>int8</code> vector dot product) and emits only the Top 8–12 L1 facts (<= 300 tokens total, including priority-1 guards) , each prefixed with <code>[F104]</code> .
L2	Full Verbatim Context	100–2,000 tokens	Remains at 0 active tokens on disk; loaded only on explicit <code>brain_expand F104</code> .

2.3 Deterministic Usage-Based Silencing & Hebbian Link Tuning

As a personal database grows over years to tens of thousands of facts, older or tangential facts can begin competing with active rules. Rather than spending LLM tokens on background summarization, XBrain uses a **100% deterministic Python plasticity loop** modeled on epigenetic methylation:

1. **The Citation Contract:** Every bot carries a simple standing rule in its charter and ticket template: *"Final status line must include `[Cited: id1, id2, ...]` for every Fact ID that materially influenced the action. Omit none."*
2. **Post-Task Deterministic Update (Zero LLM Tokens):** Each fact carries a methylation score `M_i` in `[0.0, 1.0]` (default `0.10`) and a consecutive unused-load counter `U_i`. When a task ticket closes, a pure-Python audit compares the loaded Fact IDs against the `[Cited: ...]` IDs in the bot's final status line:
 - **If Loaded AND Cited (Active Utility Confirmed):** `M_i <- max(0.0, M_i - 0.10)` and `U_i <- 0`. Simultaneously, any two chunks `(a, b)` whose facts were co-cited in the same task receive a gentle, bounded Hebbian update to their spreading-activation weight `W_ab` in `links.jsonl`: `W_ab <- min(0.75, W_ab + 0.02)` (capped at `0.75` to prevent cross-load cascades).
 - **If Loaded 5 Consecutive Times Without Citation (`U_i >= 5`):** `M_i <- min(1.0, M_i + 0.15)`.
3. **Automatic Silencing Threshold:** Any fact with `M_i > 0.70` is automatically excluded from normal trigger loading (behaving as background `_bg` memory) while remaining 100% intact and searchable via explicit query.

2.4 Cold Episodic Vault & Multi-AI History Distillation ("Sleep Spindle")

Users who have spent years working across ChatGPT, Claude, Gemini, and Grok want a way to bring all of that history into one place without drowning their active context in noise. XBrain handles multi-platform imports through a **two-stage offline distillation pipeline**:

- **Stage 1 — Cold Episodic Vault (`store/episodes.parquet`):** Raw exports from ChatGPT (`conversations.json`), Claude, Gemini, and Grok are normalized into `(episode_id, platform, timestamp, title, user_msg,`

`assistant_msg`), compressed with `zstd-19` into `episodes.parquet`, and indexed in a separate FTS5 database. This vault costs **0 active tokens** while providing 100% verbatim recall via `brain_recall_episode <id>`.

- **Stage 2 — Offline Distillation:** A background pass extracts only **four durable artifact types** from raw conversations into active chunks, tagged with `provenance_id = episode_id`:

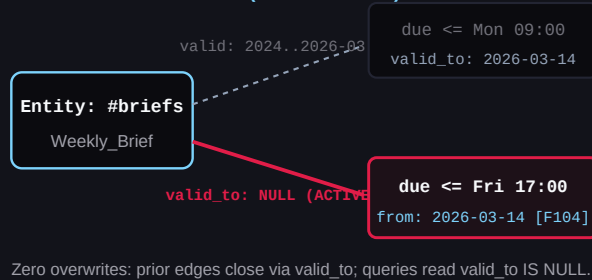
1. **Rule** — Standing preferences, operating constraints, and communication style
2. **EntityFact** — Biographical facts, people, accounts, properties, and infrastructure
3. **Decision** — Key project or life choices and the rationale for why alternatives failed
4. **OpenLoop** — Active projects, milestones, and unresolved commitments

All transient chatter and one-off boilerplate remain permanently in the cold Parquet archive.

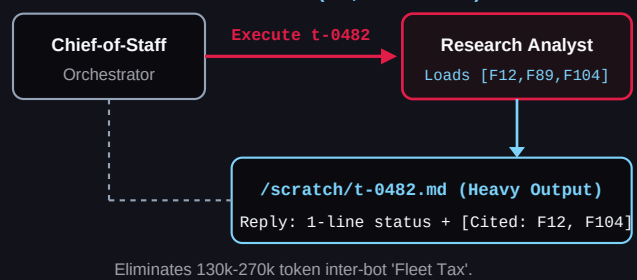
2.5 Bi-Temporal Entity Graph & Ticket-Pointer Orchestration

FIG 2.5 // BI-TEMPORAL ENTITY GRAPH (NON-DESTRUCTIVE TRUTH) & TICKET-POINTER HANDOFF

A. BI-TEMPORAL GRAPH (store/xbrain.db)



B. TICKET-POINTER PROTOCOL (< 1,000 TOKENS)



Bi-Temporal Graph inside SQLite

To prevent multi-year historical facts from conflicting when preferences or states change over time, `store/xbrain.db` maintains two derived tables rebuilt from `relations.jsonl`:

SQLITE SCHEMA // store/xbrain.db (Derived from facts/relations.jsonl)

```
CREATE TABLE entities (
  id TEXT PRIMARY KEY,
  name TEXT NOT NULL,
  type TEXT NOT NULL,
  l1_summary TEXT,
  updated_at TEXT
);

CREATE TABLE relations (
  src_id TEXT NOT NULL,
  rel_type TEXT NOT NULL,
  dst_id TEXT NOT NULL,
  valid_from TEXT NOT NULL,
  valid_to TEXT,           -- NULL = currently active state
  provenance_id TEXT,
  weight REAL DEFAULT 1.0
);
```

When a fact changes (e.g., a 2024 parameter superseded in 2026), the prior edge is closed with `valid_to` and a new edge is appended with `valid_to = NULL`. History is never overwritten, and bots query current state by default.

Ticket-Pointer Orchestration (runtime/threads.jsonl)

Instead of passing rich context or long outputs between bots in chat:

TICKET PAYLOAD // runtime/threads.jsonl

```
{
  "id": "t-0482",
  "assignee": "research_analyst",
  "goal": "Audit Q4 latency",
  "fact_ids": ["F12", "F89", "F104"],
  "scratch": "xbrain/scratch/t-0482.md",
  "status": "open"
}
```

The orchestrating bot sends only a 1-line ping: `Execute ticket t-0482`. The specialist bot pulls only the listed `fact_ids` (<100 tokens), writes heavy research or logs to `xbrain/scratch/t-0482.md`, and replies in chat with a 1–2 line summary + file pointer + `[Cited: F12, F104]`.

2.6 Portable .xbrain Containers & Proposed Future Edge Sync for Tesla Optimus

Because `xbrain/facts/` is strictly append-only JSONL and the derived `xbrain.db` SQLite index for **100,000+ distilled L1 facts occupies <50 MB**, the architecture naturally supports a **proposed future edge-sync interface** for embodied AI:

- **Lightweight Delta Sync:** A user's cloud XBrain can sync to local hardware (mobile devices, Tesla vehicles, or **Tesla Optimus**) via small, encrypted append-only JSONL delta commits (a few kilobytes per day).
- **Sub-5ms Offline Recall:** An Optimus robot in the user's home or workplace can query years of personal preferences, household rules, tool locations, and project history locally in **under 5 milliseconds** without requiring cloud LLM round-trips.

03 // Target Impact, Rollout Path & Privacy

Metric	Current Multi-Bot Behavior	Target With XBrain Patterns
Injected memory per turn	1,200–2,800+ tokens (accumulating in chat)	<= 300 tokens (60%–85% reduction)
20-turn session footprint	150k–270k+ tokens	< 15k tokens (via Scratchpads + Thread Molting)
Routine LLM wake-ups	100% of schedule ticks	>90% reduction (via 0-token local pre-checks)
Multi-bot handoff cost	130k–270k tokens ("Fleet Tax")	< 1,000 tokens via Ticket-Pointer Protocol
KV-cache prefix hit rate	Degraded by early dynamic injection	>90% (Static Prefix + Ephemeral Tail Block)
10-year personal data size	Unbounded / fragmented across platforms	< 100 MB portable .xbrain container

Phased Implementation Path & Privacy-First Telemetry

- **Phase 1:** Ship the reference `xbrain/` package and the four core tools on the Grok Bot cloud image.
- **Phase 2:** Add the `--ephemeral` tool-output flag and the `precheck_cmd` hook for Scheduled Routines.
- **Phase 3:** Add the one-click **"Molt Thread"** button and the **Universal AI History Importer** in the UI.
- **Phase 4:** Standardize the encrypted, append-only `.xbrain` delta-sync format across X, mobile, Tesla OS, and Optimus.

Privacy & Telemetry: All personal data remains on the user's cloud computer. Only four aggregate, non-PII counters are exported to `telemetry/public_health.json`: (1) average tokens injected per check, (2) check adoption rate,

(3) percentage of facts automatically silenced (`methylation > 0.70`), and (4) item counts across memory tiers (`core`, `hot_eligible`, `methyated_bg`, and `cold_episodes`).

04 // Reference Build Notes: What Is Running Today

The pattern above is not only a design sketch. This section summarizes how each primitive is implemented in the reference build: a single Python CLI (`brain/`) over an append-only JSONL store, a derived SQLite FTS5 index, and a regression gate of **283 passing tests (1 skipped, 0 failing)**. Token and percentage figures elsewhere in this paper remain **design targets** unless explicitly labeled measured.

4.1 Quiet Top-K Output (≤ 300 tokens)

`brain q <topic> --agent TAG` ranks L1 facts with FTS5 plus an int8 vector dot product (falling back to FTS5-only if the embedding library is missing) and prints at most 8–12 lines of the form `[F104] <L1 text>`, budgeted to a **300-token** ceiling, with Priority-1 guard rules always included. Everything else stays on disk: `brain expand F104` returns the L2 verbatim text and `brain recall-ids` fetches a batch by ID. Quiet mode is the default (a one-file `QUIET_DEFAULT` switch), so a bot never has to remember a flag.

4.2 Pointer-Only Tickets

A ticket is one JSON line appended to `runtime/threads.jsonl` with the assignee, goal, `fact_ids`, scratch path and status. The sending bot's entire chat message is `Execute ticket t-0482`. The receiving bot loads only the listed facts, writes bulky output to `runtime/scratch/t-0482.md`, and answers with a 1–2 line status, the file pointer and `[Cited: ...]`. Target handoff cost: **< 1,000 tokens**, versus 130k–270k when full context is re-sent.

4.3 Thread Molting

`brain molt` saves the active task state (open tickets, decisions, next step) into the brain and starts a fresh thread that re-loads only that state, so long-running work does not drag an ever-growing transcript behind it. Proposed triggers: **after 20 turns or roughly 35K tokens of thread context, whichever comes first**. Both thresholds are targets to be tuned from telemetry, not measured optima.

4.4 Flock-Guarded, Zero-Token Scheduler

The shared cloud computer has no cron daemon or user-level systemd, so scheduling is a long-running Python process, `brain_sched.py`, guarded by an `flock` so only one instance can ever run and restarted from a login hook. Each tick runs the deterministic checks as plain subprocesses (**0 LLM tokens**), appends one JSON line per check to `telemetry/sched.jsonl`, and escalates only when a check exits non-zero: `escalate.py` posts to a webhook that wakes the bot, or queues to `scratch/escalations.jsonl` until a webhook URL is configured. This is the `precheck_cmd` behavior proposed in Section 01, built outside the harness.

SCHEDULER LOG // `telemetry/sched.jsonl` (illustrative record)

```
{"ts": "2026-10-27T06:20:00-07:00", "check": "inbox_health", "exit": 0,
  "llm_tokens": 0, "escalated": false, "ms": 42}
```

4.5 L0 / L1 / L2 Codons

`codons.json` is a versioned dictionary that maps recurring entities and verbs to short symbols (for example `@User`, `!rule`, `#briefs`), so L1 lines compress to ≤ 20 tokens and L0 primers to ≤ 8 . Codons are derived data: expansion is a pure lookup, and the L2 text always stays verbatim on disk. Chunk files carry all three tiers plus the methylation score.

4.6 Methylation and Hebbian Weighting

Every fact carries a methylation score `M_i` (default 0.10) and an unused-load counter `U_i`. A fact that is loaded and cited in the closing `[Cited: ...]` line drops by 0.10; a fact loaded five times without a citation rises by 0.15; above **0.70** it moves to background (`_bg`) and stops loading by default while remaining fully searchable. Chunks co-cited in the same task strengthen their link `w_ab` by +0.02, capped at 0.75 so one hot cluster cannot cascade. The whole loop is deterministic Python run when a ticket closes: **0 LLM tokens**.

4.7 Bi-Temporal Graph and Cold Episode Vault

`relations.jsonl` is append-only. Each edge carries `valid_from`, `valid_to` and `provenance_id`; when a fact changes, the old edge is closed rather than overwritten. Current-state queries filter on `valid_to IS NULL`, and point-in-time questions ("what was the rule last March?") filter on the interval. Raw imported chat history lives in `episodes.parquet`, a cold vault costing **0 active tokens**; distilled facts point back to it through `provenance_id`.

4.8 Measured vs. Target

Measured: the reference build's regression gate (283 passing tests, 1 skipped). **Targets, not yet measured at scale:** ≤ 300 tokens injected per check, $< 1,000$ -token handoffs, $> 90\%$ fewer routine LLM wake-ups, molting at 20 turns or $\sim 35K$ tokens, and a < 100 MB container per decade of use. Usage telemetry (tokens injected per check, scheduler runs that avoided an LLM wake-up, facts silenced) is being logged in parallel so each target can be replaced with a measured number.

Thank you to the xAI and Grok Bot teams for building an incredible platform. I am happy to share any part of our live [brain/](#) reference scripts, test suite, or benchmark logs if helpful.