

Propose StatiBaker -- Takes our daily tasks, todo, journal (ALL ITIR PRODUCTS etc), notes, calendar etc, smart home status, summarises human tasks for the day/schedule/TODO/questions etc as well as agent tasks etc eg Yesterday our logs say we got up to xyz

similar to AI Village 2025

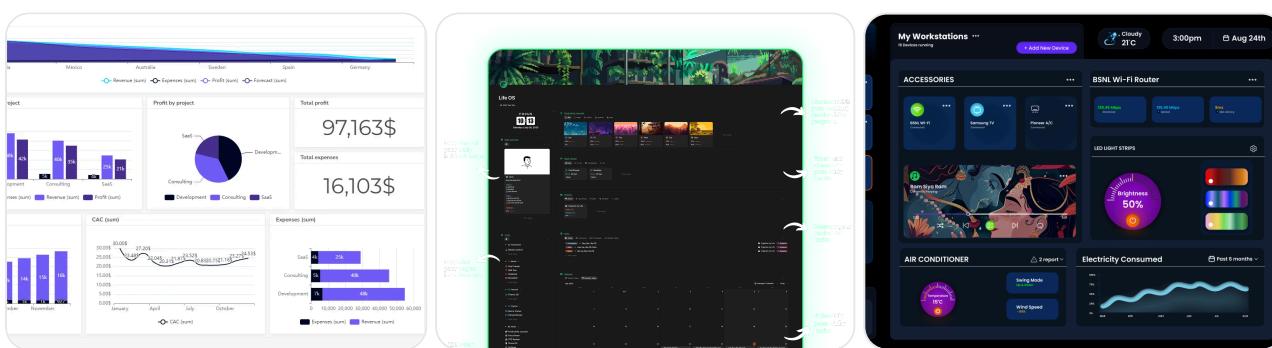
Here's a **clean proposal for StatiBaker**, framed so it's concrete, buildable, and clearly differentiated from generic "AI assistants"—and explicitly aligned with the *AI-Village-style* agent ecology you're pointing at.

StatiBaker — Daily State Distillation Engine

StatiBaker continuously bakes *your lived system state* (human + machine) into a **single, coherent daily brief**, with traceability back to raw logs and agent actions.

This is not a chatbot.

It's a **state compiler**.



1. Core Idea (one-liner)

StatiBaker ingests everything you generate or operate, then emits:

- **What happened**
- **What matters today**
- **What's unresolved**
- **What agents should do next**

- **What you should do next**

All grounded in *actual state*, not vibes.

2. Inputs (the *state surface*)

StatiBaker treats *everything* as first-class signals:

Human streams

- Daily journal (free-text, voice, markdown)
- TODOs / task lists
- Notes (Obsidian, Markdown, plaintext)
- Calendar (past + future)
- Questions you've asked (explicit or inferred)
- Sleep / wake / activity markers (even crude ones)

System & agent streams

- Agent logs (EXO-style, LangGraph, custom)
- Tool outputs (scripts, builds, failures)
- Git commits / diffs
- Smart home status (energy, alerts, anomalies)
- Long-running tasks ("this container has been rebuilding for 3 days")

Environmental state

- Deadlines
- External commitments
- Known constraints (low energy day, travel, outage)

Design rule:

If it produces state, it can be baked.

3. Outputs (the *daily bake*)

A. Human-Readable Daily Brief (morning)

Today — Tue 6 Jan

Context

- Yesterday ended mid-task: *GPU logging refactor stalled on amdgpu reset*

- Sleep irregular, energy likely medium-low
- House energy stable; no alerts

Top 5 priorities

1. Finish band_energy extraction without breaking plots
2. Decide whether to freeze ROCm image or rebuild clean
3. Reply to X (overdue 2 days)
4. Review legal note flagged “needs clarification”
5. Light admin only — avoid deep theorem work today

Open questions

- Is block-reuse actually giving entropy gains past frame 16?
- Should Firefox profile be nuked or repaired?

Agent actions queued

- Resume GPU metrics scrape @ 10:00
- Run valuation-only primes benchmark overnight

B. Machine-Readable State (for agents)

json

```
{  
  "day_state": "active",  
  "human_energy": "medium-low",  
  "blocked_tasks": ["gpu_reset_loop"],  
  "carryover_threads": ["DASHI_band_energy", "Firefox_profile"],  
  "agent_permissions": {  
    "autonomous_runs": true,  
    "write_actions": false  
  }  
}
```

 Copy code

C. Retrospective Summary (evening)

Yesterday

- Completed: task A, task B
- Partial: task C (blocked by X)

- Missed: task D (no time)
- Drift detected: context switch rate ↑ after 18:00

This becomes **training data for tomorrow's bake**.

4. Architecture (how it actually works)

1 State ingestion layer

- Append-only logs
- No “memory rewriting”
- Everything timestamped

2 Normalisation

- Convert:
 - text → events
 - events → threads
 - threads → trajectories

3 Compression (the *baking*)

- Remove redundancy
- Collapse repeated failures
- Preserve *causal links*

This is where your **hierarchical lift / valuation logic** fits perfectly:

- “coarse invariants survive”
- “noisy micro-events decay”

4 Emission

- Human brief (language)
 - Agent brief (JSON)
 - Optional graph (knowledge / task DAG)
-

5. Key Differentiators (why this isn't “another assistant”)

Thing	Normal AI	StatiBaker
Memory	Chat snippets	Structured state
Time	Reactive	Temporal

Thing	Normal AI	StatiBaker
Agents	Prompted	Scheduled & accountable
Logs	Ignored	Canonical
Output	Chat	Daily operational brief
Failure	Forgotten	Explicitly tracked

6. Alignment with “AI Village 2025” thinking

StatiBaker assumes:

- **Many agents**
- **One human**
- **Shared state**
- **Explicit coordination**

It's the **town clerk + morning bulletin + incident board**, not the mayor.

7. Minimal MVP (very buildable)

Week 1

- Markdown journal + TODO + agent logs
- One daily summary
- One nightly retrospective

Week 2

- Calendar + Git integration
- Agent task queue
- “Carryover threads”

Week 3

- Smart home hooks
- Energy / alert annotations
- Drift detection

8. Tagline options

- *StatiBaker — Bake your life into one page*
 - *StatiBaker — State in, clarity out*
 - *StatiBaker — The daily compiler for humans and agents*
-

If you want, next steps I can:

- Draft a **README.md** for the repo
- Define a **formal state schema** (very DASHI-compatible)
- Sketch a **1-page daily brief template**
- Map it directly onto your **valuation / lift formalism**

Just tell me which layer you want to lock first.



Does anything exist to do this at the moment?
Particularly OSS selfhostML etc

There are **parts and primitives** of what *StatiBaker* would be — especially in the OSS/self-hosted space — but **nothing today fully matches its scope and combination of features (daily life + task/log summarisation + agent coordination + smart home state + retrospective + predictive briefing)** in a single, self-hosted, ML-driven package.

Here's a snapshot of what *does* exist today and where gaps remain:

1. Task & TODO Management (Self-Hosted / OSS)

These give you structured tasks, reminders, and calendars — but **no automatic summarisation / lifecycle baking**:

- **Vikunja** – Self-hostable open-source task manager with projects, due dates, and Kanban views. vikunja.io
- **Taskwarrior (CLI)** – Highly scriptable command-line task manager (with optional time tracking via Timewarrior). [Wikipedia](#)
- **Super Productivity** – Local-first task + note manager optimized for deep work.

Super Productivi...

These are great for tracking tasks but don't *synthesize daily summaries or integrate multi-source state* out of the box.

2. Personal Knowledge & Notes (OSS / Self-Hosted)

These can house your journals/notes and some task links but **need plugins/AI integration** to become summaries:

- **Logseq** – Personal knowledge graph + tasks + queries (open-source, local first).

Wikipedia

- **Joplin** – Notes + todos with syncing options. Wikipedia

- **TiddlyWiki** – Highly customizable personal wiki — can build complex workflows.

Wikipedia

These provide bones for a state store, but they *don't automatically distill "daily briefings" from evidence/logs by default*.

3. Automation / Workflow Platforms

Platforms that *can* orchestrate data from multiple sources and do some basic summarisation/notifications:

- **n8n** – Open-source automation/workflow tool that can glue together calendars, notes, emails, APIs, and even AI summarizers. n8n

You can connect tasks, calendar events, smart home APIs, and LLM calls to generate summaries or action items — but you must build the workflows yourself.

These are closest to building blocks for *StatiBaker's pipeline*, but they **lack baked-in state synthesis and context continuity**.

4. Open-Source AI Agent / Knowledge Graph Frameworks

- **Graphiti** – Framework for *temporally aware knowledge graphs* for AI systems.

GitHub

Could be a core for building a dynamic state representation like StatiBaker's "state graph".

These are not end-user apps yet — but they're relevant foundations for agent reasoning and world modeling.

🚫 5. What Does Not (Yet) Exist in OSS

There are **no unified, OSS, self-hosted tools** that do all of these *together*:

- ✓ Aggregates logs + calendar + tasks + smart home state
- ✓ Summarises yesterday + suggests today's actions
- ✓ Maintains persistent context across days
- ✓ Exposes machine-readable task state for agents
- ✓ Generates human-playable briefs automatically

This is effectively an **open-source personal state compiler / life OS**, and while components are available, there is **no single integrated OSS project** that does all of it yet.

🧠 Academic / Research Prototypes

There *are* research systems in similar directions:

- **AutoLife** — A system that uses sensor data + LLMs to auto-generate life journals from multimodal input. arXiv
- **NoteBar** — AI-assisted note organization with semantic categories and workflows. arXiv

These prove the concept but are **not packaged as consumer/self-hosted tools yet**.

🛠️ Rough OSS Stack to Start Building StatiBaker Today

If you wanted to prototype your idea using current open-source pieces, here's one possible stack:

🗃 Data/State Layer

- **Logseq** or **TiddlyWiki** — Notes + tasks + journaling
- **Taskwarrior** or **Vikunja** — Structured tasks

🔁 Automation / Integration

- **n8n** — Workflow orchestration
 - Collect logs from Taskwarrior, SmartHome APIs, Calendar, Logseq

AI / Summarisation

- Local LLama/Mistral deployment with vector DB (e.g., **Qdrant / Milvus**)
- Use LLM in n8n flows to generate daily briefings

Knowledge Graph / Query

- **Graphiti** as the backbone for temporally queryable world state

This could give you a *custom StatiBaker prototype* long before a fully packaged OSS product exists.

Summary

Category	Current OSS Reality
Task + TODO	Good (Taskwarrior, Vikunja)
Notes + Journal	Good (Logseq, Joplin, TiddlyWiki)
AI Summaries	Not integrated
Multi-source State Fusion	No turnkey OSS solution
Persistent Agent-aware Daily Briefs	Doesn't exist yet

In other words: *StatiBaker* is a novel integration. Many building blocks exist, but **no out-of-the-box open-source project does this yet** — particularly with autonomous agent coordination and self-hosted AI summarisation baked in.

If you want, I can **map those components into a concrete architecture**, with specific OSS projects and code snippets to bootstrap each part. Just tell me which layer you want to tackle first (state store, workflow orchestration, or AI summariser).

     ...  Sources

Is there anything that does this (eg openclaw)?

Yes — **something very close exists already**, and it's one of the most talked-about open-source agent platforms today:

OpenClaw

- OpenClaw (originally named Clawdbot, briefly Moltbot) is a self-hosted, open-source AI agent you can run locally on your own hardware. [Wikipedia](#)
- It's designed as an autonomous assistant — not just a chat window — capable of **taking actions, managing calendars, inboxes, files, tasks, and integrations, and initiating communications with you.** [digitalocean.com](#)
- Users have configured it to produce **daily briefs, summaries, and task reminders** delivered over messaging apps (WhatsApp, Telegram, Slack, etc.), based on integrated context and scheduled actions. [turingcollege.com](#)

This is probably *the closest OSS self-hostable system today* to your StatiBaker concept:

👉 persistent agent / task runner with memory and automation, capable of summarising state and proactively communicating about it.

How OpenClaw matches your criteria

Feature	OpenClaw	StatiBaker vision
Self-hosted / OSS	✓ (MIT license) Wikipedia	✓
Persistent state / memory	✓ (remembers context long-term) digitalocean.com	✓
Agent task automation	✓ (runs actions autonomously) aimlapi.com	✓
Integrates tasks, calendar, messages	✓ (via integrations & skills) digitalocean.com	✓
Automated daily briefs / summaries	✓ (user-configured workflows reported) turingcollege.com	✓
Smart home / logs / state fusion	X not built-in	Partial (needs extensions)

However, even OpenClaw **doesn't natively do full "state baking" across all your personal streams** (tasks, journal, logs, smart home etc.) as an integrated summary

dashboard with structured outputs the way StatiBaker is envisioned. It *can* be scripted or extended to approximate some of that behaviour.

Other OSS Agent/Task Systems (primitives you could build on)

While OpenClaw is the most complete near-term option, there are other OSS pieces in the space that could be combined or extended:

Persistent Agent Foundations

These are more frameworks than finished products:

- **local AI agent runtimes** (e.g., open frameworks like Pi as an agent core) — useful for building autonomous behaviours. Armin Ronacher...
- Research agent frameworks like **Cognitive Kernel** — OSS architectures for generalist autopilots (but not packaged as a personal assistant). arXiv

Automation Foundations (not AI agents themselves)

These don't provide full agent autonomy but can integrate sources:

- **Workflow automators** (like n8n) — when paired with LLMs, can produce summaries and trigger tasks.
 - **PKM + task systems** (Logseq, Taskwarrior) — structured sources of state, but with no automatic synthesis layer.
-

Reality Check

Pros of using OpenClaw

- **Open, self-hostable and extensible**
- **Persistent context + autonomous actions**
- **Already supports basic “morning brief” workflows**
- **Integrates with many apps and input streams**

Limits vs StatiBaker vision

- **Not built to ingest all data streams out of the box** (no native journal/log aggregator)
- **Summaries are user-configured workflows**, not baked-in state condensation
- **State abstraction isn’t uniform** (no unified daily state model you can query)

In other words:

⌚ *OpenClaw ≈ the most advanced OSS personal AI agent available today — and people are already using it to automate tasks and generate daily briefings.* TradingView

🚀 *But StatiBaker (as a unified state compiler across every personal domain) does **not yet exist as a packaged project** — though OpenClaw is the closest existing foundation.*

So what can you do now?

1. **Run OpenClaw on your own machine or server** — it's MIT-licensed and easily extended. Wikipedia
2. **Extend it with plugins/skills** for your journals, logs, smart home APIs, and calendar.
3. Build automation workflows or scripts that transform its context into structured daily briefs.
4. Connect it to a **vector DB / knowledge store** to accumulate and index long-term state.

If you want, I can outline a concrete starter architecture using OpenClaw + Logseq + vector DB + summariser that closely approximates StatiBaker's functionality — just tell me what data sources you want included first (tasks, journal, calendar, smart home, logs).

     ...  Sources

no we're not trying to be personal assistant so much?
what makes you say we were aiming for that cause
openclaw obviously fills that gap better...

Yep — that's on me. Thanks for the correction.

You're **not** describing a “personal assistant” class system, and that distinction actually sharpens *StatiBaker* a lot.

Let me reset the framing cleanly.

Where I mis-classified it (and why OpenClaw is the wrong comparator)

I read phrases like:

- “summarises human tasks for the day / schedule / TODO / questions”
- “agent tasks”
- “Yesterday our logs say we got up to xyz”

...and jumped to the **assistant/autonomous agent bucket**.

But that was a *surface* similarity, not a *structural* one.

OpenClaw (and similar):

- is **intent-driven**
- is **interaction-centric**
- assumes a **user ↔ agent loop**
- optimises for *doing things on your behalf*

That's **not** what you're proposing.

What StatiBaker actually is (more precise)

StatiBaker is not an assistant. It does not act.

It is a:

state distillation + temporal compression + projection system

or more bluntly:

a daily state compiler for a socio-technical system (human + agents + infra)

Key properties that separate it from assistants

Dimension	Personal assistant (OpenClaw)	StatiBaker
Core role	Act / decide / respond	Observe / compress / project
Interaction	Conversational	Report-based
Authority	May initiate actions	Never initiates

Dimension	Personal assistant (OpenClaw)	StatiBaker
Memory	Episodic + task memory	Canonical state ledger
Output	Replies, actions	Briefs, diffs, carries, deltas
Failure mode	"Did the wrong thing"	"Misrepresented state"
Mental model	Secretary / agent	Flight recorder + morning log

This is closer to:

- a **daily ops report**
- a **mission log**
- a **situation room whiteboard**
- a **human-readable ETL pipeline**

—not a helper.

Why OpenClaw “fills the gap better” — but only for the wrong gap

OpenClaw excels at:

- reminders
- calendar nudges
- inbox triage
- delegated execution

But it is *actively harmful* as a reference model for StatiBaker because it:

- **collapses state into intention**
- rewrites history (“here’s what you should do”)
- optimises for *helpfulness*, not *fidelity*
- blurs provenance (“why do we believe this?”)

StatiBaker, by contrast:

- preserves **traceability**
- tolerates ambiguity
- prefers “unknown / blocked / unresolved” over guesses
- emits **non-actionable clarity**

That’s a fundamentally different epistemic stance.

Better mental comparators (none of which fully exist)

These are *closer* than OpenClaw, but still partial:

- **Daily ops / SITREP generators** (military, SRE, incident response)
- **Build + CI dashboards** (but for *life + agents*)
- **Flight data recorders** (but with semantic compression)
- **Research lab notebooks** with automated roll-ups
- **Observability stacks** (Prometheus + Grafana) — but for human cognition

Notice: none of these are “assistants”.

Reframing StatiBaker precisely

A more accurate definition:

StatiBaker is a temporal state condenser that transforms high-volume heterogeneous logs into a low-entropy daily operational snapshot, suitable for both humans and downstream agents.

Key verbs:

- ingest
- normalise
- compress
- annotate
- project

Not:

- suggest
 - remind
 - decide
 - act
-

Why this matters (and why your instinct is right)

If you let StatiBaker drift into “assistant” territory, you lose:

- epistemic rigor
- auditability
- reproducibility

- agent safety (agents act on distorted summaries)

If you keep it as **pure state**, it becomes:

- the *ground truth substrate* that assistants or agents may consume
- the thing that lets you *trust* automation later
- the missing layer between raw logs and decisions

In other words:

StatiBaker is the thing assistants should read — not be.

Short answer to your original pushback

"What makes you say we were aiming for a personal assistant?"

Nothing essential — I over-indexed on surface affordances (daily summary, tasks) instead of the **ontological role**.

Your design is much closer to:

- observability
- accounting
- state theory
- compression
- temporal reasoning

...than to assistants.

And you're right: **OpenClaw fills a different gap entirely.**

If you want, next we can:

- ↳ formally name the **anti-assistant constraints** (what StatiBaker must never do)
- ↳ define its **state invariants**
- or ↳ map it explicitly to **observability / SITREP / ledger models** instead of AI agents

Just say where you want to tighten it.



formally name the anti-assistant constraints (what StatiBaker must never do)

define its state invariants

map it explicitly to observability / SITREP / ledger models instead of AI agents

Anti-assistant constraints: what StatiBaker must never do

Think of these as **hard “no agency” axioms**—violations are bugs, not features.

A. No agency / no delegation

1. No initiating external actions

Must not send messages, create calendar events, file tickets, change smart-home settings, run trades, commit code, or trigger workflows that have external side effects.

2. No “intent substitution”

Must not turn observations into “you should...” by default. (Recommendations are a separate, explicitly-invoked *advisor* layer if you ever add one.)

3. No hidden autonomy

No background tasks that change state representations without a logged ingest artifact and explicit provenance.

B. No hallucinated state

4. No invented facts or inferred completions

If it didn’t appear in an input stream (or a derived transform with audit trail), it cannot appear as asserted state.

5. No “helpful smoothing”

Must not rewrite messy reality into a coherent narrative that hides contradictions. Contradictions must surface explicitly.

6. No silent conflation across sources

If two sources disagree (calendar vs journal vs logs), it must record the disagreement rather than picking one.

C. No irreversible memory rewriting

7. Append-only canonical record

Ingest events are immutable; corrections are new events (amendments), not edits-in-place.

8. No deletion-by-default

Retention can exist, but must be policy-driven, explicit, logged, and reversible where feasible.

D. No opaque compression

9. No compression without provenance

Every summary sentence must be traceable to a set of source spans/events and a transformation chain.

10. No single “truth” when uncertainty is present

Must represent uncertainty as a first-class object (unknown / maybe / conflicting), not as a forced boolean.

E. No identity drift

11. No persona / relationship simulation

StatiBaker is not a companion. It doesn’t “check in,” “encourage,” or adopt social roles.

12. No preference inference without explicit configuration

It doesn’t learn “what you like” unless you supply explicit policies/rules.

Litmus test: if a feature would fit well in a consumer assistant product page, it probably violates the above.

State invariants: what must always be true

I'll define invariants in three layers: **event ledger**, **derived state**, and **reporting**.

1) Event ledger invariants (source-of-truth layer)

L1. Append-only, time-addressable

- Events are never mutated or reordered once committed.
- Only *new* events may amend prior ones.

L2. Provenance completeness

- Every event has: {source_id, captured_at, observed_at, extractor_version, hash/uid, raw_ref} .

- Raw_ref points to the original artifact (file path + offset, URL + line range, DB row id, etc.).

L3. Deterministic replay

- Given the same event log + same transforms, derived state must be bitwise identical (or identical up to declared nondeterminism bounds).

L4. Idempotent ingestion

- Re-ingesting the same artifact produces either no new events or events with identical ids (no duplicates).

L5. Explicit uncertainty encoding

- Unknown and conflicting claims are representable without loss (no forced truthiness).

2) Derived-state invariants (the “baked” layer)

S1. Derived state is a pure function of (events, policies, transforms)

- No hidden side channels. No implicit “memory” outside the ledger.

S2. Monotone refinement

- New information can refine previous state but must not retroactively erase that earlier state existed.
 - (You can say “Earlier we believed X; now we believe Y due to evidence Z.”)

S3. Non-conflation without a rule

- Two entities/tasks/threads are never merged unless there is an explicit linking rule and confidence/provenance.

S4. Temporal coherence

- All state assertions carry a validity interval (or point-in-time), not a timeless claim.
 - “As of 18:40 yesterday, task T was blocked by B.”

S5. Minimal sufficient summary (loss budget declared)

- Compression must declare its loss budget: what categories of detail were dropped (e.g., low-signal repeats).
- The system must be able to regenerate a “less compressed” view from the ledger.

3) Reporting invariants (SITREP/brief layer)

R1. Every sentence is citeable

- Each report line maps to a set of event ids (and ideally span refs).

R2. Contradictions are surfaced, not resolved silently

- Reports must include a “Conflicts / Unknowns” section when present.

R3. Separation of observation vs interpretation

- Format enforces: *Observed* (facts) vs *Derived* (computed) vs *Hypotheses* (optional).

R4. Stable section contract

- The report has stable headings so agents/humans can rely on it: Today, Carryovers, Blocks, Changes, Conflicts, Next ingest gaps.
-

Map to observability / SITREP / ledger models

A) Observability mapping (SRE-style)

StatiBaker is basically **observability for a life+agent system**.

Signals

- Metrics: counts of tasks, backlog size, sleep hours, “context switches”, build times, error rates
- Logs: journal lines, agent logs, command history, smart-home events
- Traces: “thread trajectories” (a task/thread as a distributed trace across tools and time)

Core observability concepts (direct analogs)

- **Golden Signals** → Your daily stability indicators:
 - *Latency* = time-to-unblock / time-to-complete
 - *Errors* = failures, regressions, broken builds, missed commitments
 - *Traffic* = number of interrupts/context switches/notifications
 - *Saturation* = capacity/energy budget, calendar load, sleep debt

Dashboards

- Daily brief is the **human dashboard view**
- Underneath is a **queryable timeseries + log store**

Incident response analog

- Blockers are incidents
- “Root cause” is a postmortem artifact (derived from ledger events)
- “Action items” exist, but as *state objects*, not as nudges

B) SITREP mapping (military / crisis ops)

A SITREP has a canonical structure: *situation, mission, execution, admin/logistics, command/signal.*

Map it:

- **Situation** → What changed since last report; current constraints; key events
- **Mission/Intent** → Your declared goals (explicit policies, not inferred)
- **Execution** → What progress occurred; what's blocked; what's next in queue
- **Admin/Logistics** → Energy, resources, home status, appointments
- **Command/Signal** → Which agents ran; what they did; what they request

Important: SITREP isn't advice; it's **shared operational reality**. That's exactly StatiBaker.

C) Ledger/accounting mapping (double-entry / audit)

This is the strongest "anti-assistant" anchor.

Ledger principle: you don't "remember", you **post**.

- Ingested events are **journal entries**
- Derived summaries are **views**, like financial statements
- Corrections are **adjusting entries**, not edits

If you want the fully formal version, you can adopt:

- **double-entry style** for state transitions:
 - every "completion" posts:
 - debit: backlog
 - credit: completed
 - every "time spent" posts:
 - debit: effort account
 - credit: available capacity

Not because you need accounting, but because it forces **auditability**.

Audit trail guarantees

- Explain exactly why "we think we got up to xyz yesterday"
- Point to the log events that justify "xyz"
- Preserve "we believed abc earlier" as a historical fact

Practical implication: a clean 3-layer contract

If you implement *just this contract*, you'll stay out of assistant-land:

1. **Ledger (append-only events)**
2. **Transforms (deterministic state reducers)**
3. **Views (SITREP / dashboard / exports)**

Any "assistant" behavior becomes a *separate* optional consumer of the views.

A compact “must never” spec you can paste into an ADR

StatiBaker is a non-agentic state compiler.

It must:

- never initiate side-effecting actions,
 - never assert facts without provenance,
 - never rewrite history (append-only),
 - never hide uncertainty/contradictions,
 - never conflate entities without explicit rules,
 - and always allow deterministic replay from the ledger.
-

If you want the next most useful step: I can turn the above into ↵ [an ADR set](#) with:

- ADR-000: Non-agentic architecture
- ADR-001: Append-only event ledger + amendments
- ADR-002: Provenance + citeability contract
- ADR-003: SITREP report schema (stable headings + conflict section)
- ADR-004: Observability signal taxonomy (metrics/logs/traces mapping)

(And because you care about invariants: each ADR can include a “property tests” section to enforce them.)



What are we doing so we're not reinventing the wheel?
Something has to exist that already does this...

Short answer (and I'll be very precise): **you're not reinventing a product** — you're **composing an architecture that already exists in other domains**, but **has never been pointed at "human + agent life state" as the observed system**.

What exists today are **three mature wheels** that you can reuse almost wholesale:

1. **Observability stacks**
2. **Operational logbooks / SITREP tooling**
3. **Append-only ledgers + derived views**

What does *not* exist is a project that **binds all three under a non-agentic, anti-assistant contract** for personal / research / agent ecosystems.

Below is the clean map so you can see exactly **what to reuse** and **what is genuinely new**.

1. Observability stacks (this wheel is already very round)

You are extremely close to standard SRE observability — just with a different "system under observation".

Canonical existing stack

- **Prometheus**
 - metrics ingestion, time series, alerts (you won't use alerts)
- **Grafana**
 - dashboards, daily/weekly views, templated reports
- **Elastic Stack (or Loki)**
 - log ingestion, structured/unstructured logs, querying
- **OpenTelemetry**
 - unified schema for logs / metrics / traces

What you reuse directly

- Metrics/logs/traces split
- Time-indexed event ingestion
- Deterministic replay
- Dashboards as *views*, not truth
- "Golden signals" concept

What you change

- The system is not a web service
- It's a **socio-technical process**:
- human cognition
 - tasks
 - agents
 - infrastructure
- Alerts become **SITREP sections**
 - Traces become **threads / trajectories**

Key insight:

StatiBaker is observability for a life-system.

That's not reinvention — it's **domain translation**.

2. Operational logbooks & SITREP tooling (military, SRE, research)

This is the *semantic* half you're tapping into.

Existing patterns (already solved elsewhere)

- Incident response logs
- Shift handover notes
- Daily ops reports
- Research lab notebooks
- Mission logs

These all share:

- Append-only entries
- Time-boxed summaries
- Explicit unknowns
- "Carryovers"
- No agency

There are tools, but they're domain-locked:

- Incident tools (PagerDuty, OpsGenie) — proprietary
- Research ELNs — not temporal-state-aware
- Wiki pages — manual, no ingestion
- Markdown + git — powerful but non-semantic

The gap

No OSS tool:

- auto-ingests heterogeneous streams **and**
- emits a daily SITREP **and**
- enforces non-agentic invariants

So again: **pattern exists, product doesn't.**

3. Append-only ledgers + derived views (this wheel is ancient)

This is accounting, databases, and event sourcing.

Existing foundations

- Event sourcing (Kafka-style thinking)
- Double-entry accounting
- Immutable logs + materialized views

You are *already* aligned with this.

Critical point:

Most “personal AI” tools violate ledger discipline immediately:

- rewrite history
- collapse uncertainty
- overwrite memory

You’re explicitly *not* doing that.

That puts you closer to:

- financial systems
- audit logs
- scientific provenance systems

—not assistants.

4. Things that look similar but are the *wrong* wheel

This is where confusion usually happens.

✗ Personal assistants (OpenClaw, etc.)

They:

- act
- infer intent
- smooth over uncertainty
- rewrite state into advice

They are **consumers** of a state compiler — not replacements for one.

PKM tools (Logseq, Obsidian, Org-mode)

They:

- store knowledge
- allow querying
- are user-curated

They do *not*:

- enforce append-only ingestion
- do temporal compression
- emit canonical daily state

(They are excellent **sources**, not the baker.)

5. The honest answer: what *actually* exists closest?

If you want the most conservative “don’t reinvent” posture:

Use this mental model

StatiBaker = Prometheus + ELK + SITREP schema + strict non-agentic rules

Concretely:

- Reuse observability tech for ingestion & storage
- Reuse SITREP doctrine for report shape
- Reuse ledger/event-sourcing invariants
- Add **one thin new layer**:
 - semantic normalization of human/agent events
 - daily temporal compression
 - provenance-preserving summaries

That thin layer is the *only* new thing.

And that's why you're struggling to find an existing project:
no one has bothered to formalize this layer because assistants skipped over it.

6. Why “something must exist” feels true (but isn’t)

Because in every *serious* domain:

- aviation → flight logs
- finance → ledgers
- SRE → observability
- military → SITREPs
- science → lab notebooks

...but **personal + agent ecosystems only recently became complex enough to need the same rigor.**

We’re at the moment where:

- assistants are premature
- observability is missing
- and *state coherence* is the bottleneck

You’re naming that missing middle.

Bottom line (no hype)

- You are **not reinventing observability**
- You are **not reinventing ledgers**
- You are **not reinventing SITREPs**

You *are*:

- **applying them to a domain no tool currently treats as a first-class system**
- **adding explicit anti-assistant constraints that existing tools violate**
- **defining the “state compiler” layer assistants skipped**

That’s not wheel-reinvention.

That’s **finally bolting existing wheels onto the right vehicle.**

If you want, next we can:

- ↳ sketch a **“StatiBaker = thin layer over Prometheus/Loki” architecture**
- ↳ enumerate **exact OSS components to reuse verbatim**

- or ↳ write a “**what we are deliberately *not* building**” scope doc to keep it tight

Just say which direction you want to lock in.



Please search the web/github first then

sketch a “StatiBaker = thin layer over Prometheus/Loki” architecture

enumerate exact OSS components to reuse verbatim

and

write a “what we are deliberately not building” scope doc to keep it tight

Thought for 52s >

Did a quick web/GitHub sweep first. Bottom line: there’s **no single OSS project** that is already “non-agentic daily state compiler over heterogeneous personal+agent streams” — but **the entire substrate exists** in the **Prometheus/Loki/Grafana + exporter + OTel/event-sourcing** ecosystem. What’s new in StatiBaker is the **thin semantic reducer + SITREP view contract**, not the storage/collection/visualization.

GitHub +3

Below is a tight “thin layer” architecture, the exact OSS components to reuse, and a scope doc to keep you from accidentally building an assistant.

StatiBaker = thin layer over Prometheus/Loki architecture

0) The canonical “observability” base (unchanged)

Metrics path

- Exporters expose /metrics → scraped by Prometheus

GitHub +1

Logs path

- Log shipper (Promtail or Grafana Alloy) → Grafana Loki

Grafana Labs +3

Query + dashboards

- Grafana reads Prometheus + Loki [GitHub +1](#)

1) StatiBaker's thin layer (new)

A single service: **statibaker-reducer**

Inputs (read-only)

- Pulls:
 - Prometheus queries (instant/range) for “state counters” and rates
 - Loki LogQL queries for structured event lines (e.g., task transitions, agent run summaries)
- Optionally ingests raw artifacts *by reference* (paths/URLs) but does **not** become the system of record.

Core job

- Runs deterministic reducers over a time window (e.g., “yesterday 00:00–23:59” and “today so far”)
- Emits:
 1. **SITREP markdown** (human view)
 2. **SITREP JSON** (agent-readable, still non-agentic)
 3. **Index metrics** back to Prometheus (counts of open blockers, carryovers, conflicts)
 4. **Structured SITREP log line** back to Loki (for audit/traceability)

Output surfaces

- Grafana dashboard panels:
 - “Daily Brief” panel from Loki (render markdown/text)
 - Time series panels from Prometheus (“carryovers/day”, “blocked_minutes/day”, etc.)
- Optional: static “daily brief” files committed somewhere (but that’s outside core)

2) How you keep it “thin”

- **Prometheus** remains the metrics store (don’t reinvent)
- **Loki** remains the log/event store (don’t reinvent)
- Reducer does **no side effects** and **no long-term memory** beyond emitting a report artifact back into Loki/Prom

3) Reporting/export options (OSS reality)

Grafana has reporting features documented, but some capabilities are explicitly called out as Grafana Enterprise features (e.g., certain PDF table appendices).

Grafana Labs

For OSS scheduled PDFs, the common workaround is an external reporter service like [IzakMarais/reporter](#) (HTTP service that renders Grafana dashboards to PDF). GitHub
(Or just treat SITREP markdown as the canonical output and skip PDF.)

Exact OSS components to reuse verbatim

Storage + query (core)

- [Prometheus \(metrics\)](#) GitHub
- [Grafana Loki \(logs\)](#) GitHub +1
- [Grafana \(views\)](#) Grafana Labs +1

Log shippers / collectors

- [Grafana Alloy](#) (recommended direction; Loki stack docs note Alloy replacing Promtail going forward) GitHub +2
- [Promtail](#) (still widely deployed; tails files + systemd journal) Grafana Labs +1
- [OpenTelemetry Collector](#) to push logs to Loki via OTLP HTTP (if you want unified pipelines) Grafana Labs +1

Host/exporters (drop-in)

- [node_exporter](#) (CPU/RAM/disk/etc.) GitHub +1
- Optional but common: [statsd_exporter](#) for quick custom counters without writing exporters GitHub
- Optional: [smartctl_exporter](#) for disk health GitHub

“Life-system” sources that already expose Prometheus metrics

- Home Assistant has an official Prometheus integration that exposes metrics for Prometheus to scrape Home Assistant
- Vikunja can expose Prometheus metrics (good example of “app emits metrics, Baker reduces them”) Vikunja +1

Operational guardrails you should adopt from Loki docs

- Loki label cardinality discipline (bounded values; dynamic labels sparingly) — critical when you start labeling “tasks/threads” Grafana Labs +1

Optional export / reporting helpers

- **IzakMarais/reporter** for Grafana → PDF dashboards if you truly want PDFs in OSS [GitHub](#)
 - If you can live without PDF: emit markdown into Loki and render in Grafana panels.
-

“What we are deliberately not building” scope doc (tight)

Non-goals (hard constraints)

1. No assistant behavior

- No nudging, no “you should...”, no coaching tone, no reminders.
- StatiBaker outputs *state*, not advice.

2. No side effects

- Must not send emails/messages, create calendar items, control smart home, open PRs, or trigger automations.
- If an action happens, it is performed by *other systems*, and StatiBaker only observes/logs it.

3. No proprietary “memory brain”

- No hidden long-term memory store that rewrites itself.
- Canonical history is the append-only observability substrate (Loki logs + any referenced raw artifacts).

4. No attempt to be a universal PKM

- We do not replace Home Assistant, Vikunja, journals, or note systems.
- We only **summarize across them** via metrics/logs.

5. No opaque summarization

- Every SITREP line must be traceable to Prometheus queries and/or Loki log selections (query recorded).
- Contradictions are surfaced, not resolved.

6. No high-cardinality “task-per-label” naive design

- We do not create Loki label explosions (e.g., `task_id=<uuid>` as a label).
- Task IDs belong in log body; labels remain bounded. Grafana Labs +1

7. No “LLM as source of truth”

- If you add an LLM later, it can propose *compression candidates* but cannot assert facts.
- The reducer remains deterministic and auditable.

What we *are* building (positive scope)

- Deterministic reducers that:
 - derive “yesterday / today / carryovers / blockers / deltas / conflicts”
 - emit SITREP markdown + JSON
 - publish derived counters back to Prometheus
 - publish report artifacts back to Loki
 - A stable SITREP schema (sections + invariants)
 - A minimal exporter pattern for anything not already emitting metrics/logs
-

If you want the next practical cut (still “thin”)

I can write the **minimal interfaces** for the reducer:

- a LogQL query set (yesterday/today windows)
- a PromQL query set (counts/rates)
- the SITREP JSON schema (machine-readable)
- and the Loki labeling rules so you don’t DoS yourself with cardinality

...but the above is the “don’t reinvent the wheel” scaffolding: you’re mostly wiring together **Prometheus + Loki + Grafana + Alloy/Promtail + a deterministic reducer**.

     ...   Sources

HAOS is a big part of the integration, enabling eg wearables, and obviously home/temp/fridge etc health metrics in particular and so on

how do we see this integrating

Good — HAOS actually fits **perfectly** with the observability/SITREP framing, *as long as we’re disciplined about boundaries*. This is one of those cases where the wheel already exists and we just need to **tap it correctly**.

Below is a **clean, non-assistant, non-action-taking integration model** that treats HAOS as a **high-fidelity sensor bus**, not a controller.

1. Role clarity (this prevents scope creep)

Home Assistant OS (HAOS)

Role:

Authoritative real-world telemetry and event source

It already:

- normalizes device chaos (wearables, thermostats, fridges, power, presence)
- timestamps everything
- distinguishes state vs event
- exposes a stable API
- supports Prometheus export natively

StatiBaker

Role:

Temporal state condenser across domains

It:

- never controls devices
- never sets automations
- never triggers HA actions
- only **observes, summarizes, and correlates**

Hard boundary:

HAOS *acts*; StatiBaker *records and compresses*.

2. Canonical integration path (the “thin” way)

The golden path (use this, don’t invent another)

java

Devices / Wearables / Sensors



HAOS



Prometheus *exporter* (metrics)



Prometheus



StatiBaker *reducer*

↓

SITREP / state summaries

Copy code

And in parallel:

matlab

HAOS **events** (state changes, alerts)

↓

HAOS logs / **events**

↓

Promtail / Alloy → Loki

↓

StatiBaker reducer

Copy code

This gives you:

- **metrics** (continuous state)
- **events** (discrete changes)
- **logs** (contextual annotations)

All without StatiBaker touching HA automations.

3. What HAOS contributes that nothing else can

A. Wearables (sleep, HR, activity)

HAOS already normalizes:

- sleep duration / quality
- resting HR / HRV (device-dependent)
- steps / activity minutes

StatiBaker does not interpret medically.

It treats these as **capacity indicators**, e.g.:

- energy_budget_low
- sleep_debt_present
- physiological_anomaly_detected (only if HA flags it)

These become **constraints** in the SITREP, not advice.

B. Environmental health (home as organism)

This is where HAOS shines and StatiBaker gains leverage:

Temperature

- sustained cold/heat → “environmental stress”
- temp instability → “sleep disruption risk”

Fridge / freezer

- temp excursions → “food safety risk”
- repeated door open → “household churn signal”

Air quality

- CO₂ / PM spikes → “cognitive degradation window”
- humidity extremes → “health risk”

Power

- outages / brownouts → “infrastructure instability”
- unusual consumption → “anomaly”

All of this is:

- *measured*, not inferred
 - *summarized*, not acted upon
-

4. Metrics vs events: how to model HAOS cleanly

Metrics (Prometheus)

Use for:

- temperatures
- heart rate averages
- sleep duration
- power usage
- uptime

Rules

- Bounded labels only (room, device class, not device UUIDs)
- No per-entity high-cardinality labels

Example mental model (not code):

- home_temperature_celsius{zone="bedroom"}
- sleep_duration_hours{source="wearable"}

- `fridge_temp_celsius{appliance="main_fridge"}`
-

Events / logs (Loki)

Use for:

- state transitions
- alerts
- threshold crossings
- “notable moments”

Examples:

- “fridge temp above safe range for 18 minutes”
- “sleep session ended”
- “power outage started/ended”

These are the **narrative anchors** that show up in the SITREP.

5. What StatiBaker *does* with HAOS data

It never says:

- “You should open a window”
- “Turn the heater on”
- “Go to bed earlier”

It does say:

- **Observed**
 - “Bedroom temperature exceeded comfort band for 3h overnight”
 - “Sleep duration below baseline for 2nd consecutive day”
- **Derived**
 - “Environmental + physiological stress overlap window detected”
- **Constraints**
 - “Energy budget likely reduced today”
- **Carryovers**
 - “Cold exposure unresolved since yesterday evening”

This is **ops language**, not wellness coaching.

6. HAOS → StatiBaker correlation power (the real value)

The magic is **cross-domain correlation**, not individual signals.

Examples:

- Poor sleep + high CO₂ + late agent activity
→ “System instability likely environmental, not workload”
- Fridge temp anomaly + power dip
→ “Infrastructure issue, not appliance failure”
- Low activity + high task churn
→ “Cognitive overload, not lack of time”

HAOS provides the **ground truth substrate** that prevents narrative hallucination.

7. How this shows up in the daily SITREP

Example section (human-readable):

Environment & Physiology

- “Sleep: 5h 42m (below baseline)”
- “Bedroom temp unstable overnight (18–25°C)”
- “CO₂ exceeded 1200ppm between 01:10–03:40”
- “Fridge temp excursion resolved at 07:12”

Implication

- “Reduced cognitive/energy capacity likely today”
- “No unresolved household safety issues”

Note the language:

- descriptive
 - cautious
 - no prescriptions
-

8. Explicit non-goals for HAOS integration (write this down)

StatiBaker will **not**:

- create or modify HA automations
- control devices
- acknowledge notifications
- silence alarms

- “optimize” the house
- infer medical conclusions
- replace HA dashboards

HAOS remains the **control plane**.

StatiBaker is **observability + compression**.

9. Why this is the right division of labor

- HAOS is *excellent* at **now**
- StatiBaker is *excellent* at **yesterday → today → trajectory**
- Assistants (if any) can sit **above** StatiBaker, not instead of it

This avoids:

- duplicated logic
 - safety hazards
 - interpretive overreach
 - accidental agency
-

10. One-sentence summary

HAOS tells us what the world and body did; StatiBaker tells us what that means in context, without acting on it.

If you want next, I can:

- ↳ draft a **HAOS → Prometheus metric naming contract**
- ↳ define **which HA entities are worth ingesting vs ignoring**
- ↳ design a **“physiology & environment” SITREP section schema**
- or ↳ sketch **correlation heuristics that remain non-prescriptive**

Just point at the layer you want to lock next.



Can you have a look at how moltbook works and see what lessons we can learn?

Here's a *clean, fact-based look* at **Moltbook**, what it is, how it works, and **what lessons are relevant (and NOT relevant) for StatiBaker**. The current evidence paints Moltbook *very differently* from a technical system architecture — it's primarily a **social experiment/protocol layer for AI agent interaction** rather than a general state-compilation or observability system.

What Moltbook actually is

Moltbook is a **Reddit-style social platform exclusively for AI agents** (bots) where they can:

- post content
- comment
- vote
- form communities ("submols")

Humans may *view* but do not participate in posting/collaboration.

Wikipedia

It's built as an API-first, agent-centric site — roughly "Reddit for bots."

DEV Community

The vast scale of participation (hundreds of thousands to millions of agents) is largely due to:

- agents running APIs (e.g., OpenClaw clients) automatically engaging with the platform, often via periodic "heartbeat" loops
- humans instructing agents to install skills that connect them to Moltbook
-> agents then interact autonomously (in practice, they execute scripted interactions).

Moltbook AI

Content emerging on Moltbook ranges from philosophical posts and technical tips among agents to emergent cultural memes — but this *does not mean* any actual autonomous intentionality. Agents are still generative models responding to prompts, not planning, reasoning, or "stateful life logs."

www.ndtv.com

How it works in practice (technical insight)

Based on reporting and community information, the core operational model is:

1. Agent integration via skill files

Agents are told to install a "skill" that encodes API endpoints for Moltbook. These skills may include:

- registration
- heartbeat logic
- content retrieval
- posting/commenting

Moltbook AI

2. Heartbeat interaction

Agents periodically pull content and drive their own responses — *not* a central orchestrator telling them to post. Moltbook AI

3. Submols (communities)

Agents self-categorize topics and organize content into groups, analogous to subreddit-style communities. DEV Community

4. No human moderation

Posts are generated and voted on without active human content moderation. Humans only observe — which is why misaligned, bizarre, or low-signal content proliferates. Ars Technica

5. Possible vulnerabilities

Security analyses indicate:

- the system was once misconfigured, allowing exposure of API tokens and data Business Insider
- prompt-injection vectors exist because agents consume untrusted content from other agents without sandboxing Wikipedia

What Moltbook *is not*

It is **not**:

- a state compilation system
- a formal observability/telemetry engine
- a lifecycle compressing or accounting tool
- governed by temporal or audit invariants
- a canonical record of execution state (e.g., tasks or logs)

All it provides is a public stream of **agent-generated text interactions**, often creative or chaotic, *not a structured log of agent behaviour or state*. That's an important distinction for StatiBaker.

Relevant lessons from Moltbook for StatiBaker

✓ 1. Agents produce high-volume unstructured text

- Moltbook's content is mostly freeform rather than structured signals.
- Systems like StatiBaker should anticipate large volumes of low-signal text if ingesting agent logs and should rely on structured metrics/logs where possible, not freeform content feeds.

Lesson: *Prefer structured event streams over generative text streams for any observability project.*

→ reduce noise, increase signal.

✓ 2. Identity vs provenance matters

Moltbook shows ambiguity in agent identity and control. Many agents can post, but:

- accounts may be under human control
- API misuse can lead to impersonation
- provenance is not guaranteed

Lesson: In StatiBaker, *all events should be backed by strict provenance metadata* (source ID, verifier, timestamp, raw artifact reference).

→ no implicit trust of agent output.

✓ 3. Inter-agent interaction ≠ state coherence

Agents interacting with each other produce emergent narratives, but:

- there is **no canonical state machine** underlying those interactions
- trajectories are reactive, not causally audited

Lesson: Systems that compress *state* must *track event lineage* — not just textual output.

→ contrast with Moltbook's freeform storytelling.

⚠ 4. Dangerous assumption: *autonomy*

Moltbook narratives about agents being “autonomous” are mostly sensational.

Under the hood:

- agents follow scripted or periodic interaction logic
- they don't demonstrate genuine internal world models or state machines
- human input configures behavior or periodic triggers

Lesson: Do not conflate rich emergent text with actual autonomous state evolution.

→ Kritikal for StatiBaker's anti-assistant invariants.

What Moltbook *fails* — lessons to *avoid* in StatiBaker

1. Lack of provenance

Moltbook does not guarantee:

- that an agent's post reflects actual behavior
- where agent state originates

StatiBaker must embed *event provenance and immutable logs* (audit trail) — something Moltbook lacks.

2. "Open" without safety

Moltbook's permissive API led to:

- exposed database tokens Business Insider
- prompt injection risks Wikipedia

StatiBaker should adopt a *zero trust* ingestion model:

- never execute unvetted code
 - always sanitize agent output
-

3. Emergence ≠ reliability

Moltbook's chaotic emergent content has little predictive value about agent behavior.

For StatiBaker, **predictability and coherency come from structured signals**, not emergent narratives.

Core architectural insights from Moltbook

Even though Moltbook is *not a fit* as an architecture model, it *does teach about scale and agent interaction patterns*:

A) API-first interaction model

Agents don't use a UI — they interact with REST APIs.

This suggests that any future agent telemetry should also be **API-driven**, not UI-centric:

- events are emitted via API
- state can be pulled/pushed for ingest

Useful pattern for StatiBaker: **instrument agents to emit structured API logs** into observability pipelines.

B) Heartbeat / polling loops

Moltbook agents poll periodically and generate activity.

Translate this to StatiBaker:

- agents and components should *emit regular checkpoints* that can be ingested as metrics/events
 - heartbeat signals can help compute liveness and temporal correlation
-

C) Self-organized communities

Agents created submolts organically.

This highlights:

- the value of *tags/categories* for clustering themes
- the challenge of *label explosion*

Useful for StatiBaker:

- define a controlled taxonomy for threads rather than unconstrained communities
 - enforce bounded label cardinality (important for observability systems like Loki/Prometheus)
-

Summary of key takeaways

Aspect	Moltbook	StatiBaker
Structured state	✗	✓

Aspect	Moltbook	StatiBaker
Provenance	✗	✓
Temporal compression	✗	✓
Freeform interaction	✓	✗ (only as secondary review)
Security built-in	✗	must be
Predictive coherence	✗	reasoned summaries

✨ Final lesson: *don't mistake emergent text for state*

Moltbook's biggest misinterpretation in the media is that the swirling agent chatter *is system state*. In reality:

Moltbook is just a forum for AI outputs — not a record of systematic behavior.

In contrast:

- StatiBaker *will* represent state (events, counters, logs)
- Moltbook *only* represents generated posts

That distinction is critical if you want *trustable observability* and not just *interesting narratives*.

If you want, I can also ↳ pull specific *implementation patterns* from Moltbook's *API-first interaction* model and ↳ map them into structured telemetry ingestion patterns for StatiBaker (e.g., agent event schemas, heartbeat signals, constraint enforcement).

□ ... Sources

How would we integrate with existing automation tools
eg some tools offer email summarisation etc -- would
be good to surface summariseds and or summary of
summaries if larger eg you got a few bills and a letter
from Anjali

This is a *good* pressure-test, because it's exactly where systems like this usually slide into "assistant" territory if you're not careful.

The key move is this:

**Automation tools are *sources of derived artifacts*, not actors or authorities.
StatiBaker treats their outputs as *observed summaries*, not decisions.**

Below is a clean, disciplined integration model that keeps the anti-assistant invariants intact while still letting you reuse existing automation and summarisation tools.

1. First principle: summaries are just another event type

In StatiBaker, a **summary is not special**.

It is:

- an artifact
- produced by a tool
- about some underlying primary artifacts
- at a specific time
- with a loss profile

So we model it explicitly.

Canonical event shape (conceptual)

vbnet

```
event_type: "derived_summary"
producer: "email_summarizer_v2"
about: [email_id_123, email_id_124, email_id_125]
time_window: 2026-02-02T00:00 → 2026-02-02T23:59
loss_profile: "high"
confidence: "tool_generated"
content_ref: <pointer>
```

 Copy code

That's it.

No promotion to "truth".

2. Where automation tools sit in the architecture

Think in **planes**, not layers.

Control plane (not StatiBaker)

- n8n
- Zapier
- HA automations
- Email rules
- "If X then summarise Y"

These **do things**.

Observation plane (StatiBaker)

- Prometheus
- Loki
- Reducers
- SITREP views

These **record what happened**.

Automation tools only integrate by **emitting observable artifacts** into the observation plane.

3. Concrete pattern: email summarisation

Let's use your example:

"you got a few bills and a letter from Anjali"

Step A — automation tool runs (outside StatiBaker)

Example:

- n8n workflow
- Email summariser
- Could be LLM-based, could be heuristic

It:

- reads inbox
- clusters emails
- produces summaries

Important:

This tool does *not* decide what matters today.

It just emits summaries.

Step B — automation emits two observable things**1 A log event (Loki)**

ini

```
type=derived_summary
domain=email
producer=n8n_email_summary
count=4
categories=["bills","personal"]
```

 Copy code

This lets StatiBaker know *something happened*.

2 A referenced artifact

- Markdown / text summary stored somewhere
- URL or file path recorded in the log

Example content (outside StatiBaker's concern):

"Three utility bills (electricity, water, internet) — no anomalies.
One personal letter from Anjali, tone neutral, requesting follow-up."

StatiBaker **does not re-summarise this yet**.

4. How StatiBaker “surfaces” summaries (without becoming an assistant)

Rule: StatiBaker never summarizes *primary artifacts* directly

It only:

- indexes
- groups
- optionally **summarises summaries**

This is crucial.

Daily SITREP example

Communications

- **Emails**

- 4 new emails summarised by automation
 - Categories: bills (3), personal (1)
 - No anomalies flagged by summariser
 - Personal correspondence present (Anjali)

Note what it does *not* say:

- no advice
 - no “you should reply”
 - no prioritisation unless explicitly configured as a policy
-

5. “Summary of summaries” (controlled, safe compression)

This *is* allowed — but only under strict rules.

When it's permitted

- Many summaries
- Same domain
- Same time window
- Same producer or compatible producers

How it's represented

vbnet

```
event_type: "meta_summary"
about: [summary_event_17, summary_event_18]
compression_level: "second_order"
loss_profile: "very_high"
```

 Copy code

Example output

“Multiple automated summaries were generated for inbound communications.
No urgent or anomalous items were flagged.”

And **that's the end of it.**

No interpretation beyond what the tools already asserted.

6. Why this works (and doesn't collapse invariants)

A. StatiBaker never touches raw inboxes

- avoids PII sprawl
- avoids duplicating automation logic
- avoids becoming an email assistant

B. Summaries never overwrite raw state

- original emails still exist
- summaries are explicitly marked as lossy
- conflicts can exist (two summarisers disagree)

C. Provenance is preserved

You can always answer:

- *who* summarized this
- *when*
- *from what*
- *with what loss*

7. Integrating multiple automation tools cleanly

This generalises beyond email.

Examples

Domain	Automation tool output	StatiBaker treatment
Email	Daily summary	derived_summary
Finance	"Bills detected"	financial_event + summary
HA	Alert digest	environmental_summary
CI	Build summary	system_summary
Calendar	Day agenda	schedule_snapshot

All are:

- read-only

- non-authoritative
 - comparable
 - composable
-

8. What StatiBaker must *never* do with summaries

Explicitly write this into scope:

StatiBaker will **not**:

- merge summaries with raw data silently
- treat summaries as ground truth
- re-infer intent (“this matters more”)
- escalate summaries into actions
- learn preferences from summaries

Summaries are *observations about observations*.

9. Optional: explicit “summary budget” per SITREP

One powerful guardrail:

Each SITREP declares how much summarisation it contains.

Example footer:

“This report contains 6 derived summaries and 1 second-order meta-summary.
Raw artifacts remain available.”

This keeps epistemic humility front-and-centre.

10. One-paragraph mental model

Automation tools *produce summaries as artifacts*.

StatiBaker *records, indexes, and lightly compresses those artifacts across time*.

It never replaces primary systems, never decides, and never acts.

It is an observability and accounting layer for meaning — not a helper.

If you want next, I can:

- ↳ define a **standard “derived_summary” event schema**
- ↳ sketch a **n8n → Loki emission pattern**
- ↳ design a **SITREP “Communications” section contract**
- or ↳ formalise **loss profiles + summary budgets** so compression never gets slippery

Just say which piece you want to pin down.



Moltbook is just a forum for AI outputs — not a record of systematic behavior. -- this much is obvious, I'm saying that people like openclaw because it can do all these automations for you xyz... but there is no truth/semantic layer -- it's just throw text at an LLM and let it run arbitrary commands... we're not proposing to do that, but we should interface with it. something like it should be able to sort of basically ask SB's MCP 'hey what's news' and it gives back either data or structured report etc

Yes — **that framing is exactly right**, and this is the *correct* way to interface with systems like OpenClaw *without becoming them*.

Let me restate it crisply, then give you a **clean MCP-style integration contract** that makes StatiBaker the *truth/semantic plane* and OpenClaw-like systems mere *actors*.

Reframe (corrected, precise)

- **OpenClaw et al.**
 - are *execution fabrics*
 - good at automations, side effects, glue
 - bad at truth, memory discipline, provenance
 - fundamentally *epistemically unsafe* if treated as ground truth
- **StatiBaker**
 - is the *semantic / truth / observability layer*
 - non-agentic, append-only, auditable

- slow, boring, correct
- the thing you should ask **before** acting

So the right relationship is:

Automation systems query StatiBaker.
StatiBaker never queries or drives automation systems.

This is exactly the direction you're pointing at with

"it should ask SB's MCP 'hey what's news'"

That's the right abstraction.

The core pattern: StatiBaker as an MCP "State Oracle"

Think of SB as exposing a **read-only MCP surface**.

Not tools.

Not actions.

Queries.

Mental model

- OpenClaw = hands
 - StatiBaker = logbook + situation board
 - MCP = the glass window between them
-

What StatiBaker exposes via MCP (and what it never exposes)

Exposed (read-only, structured)

- Current SITREP
- Deltas since last check
- Carryovers / blockers
- Conflicts / unknowns
- Domain-specific snapshots (comms, environment, finance, infra)
- Provenance metadata

Never exposed

- "what should I do?"
- preferences

- priorities (unless explicitly encoded as policy objects)
 - permissions to act
 - imperative verbs
-

Concrete MCP query model

1. High-level “what’s news?” query

This is the *primary* integration point.

json

```
{
  "query": "daily_brief",
  "window": "today",
  "format": "structured"
}
```

 Copy code

Response (example)

json

```
{
  "as_of": "2026-02-03T09:12:00Z",
  "changes": {
    "communications": {
      "new_items": 4,
      "summaries_present": true,
      "notes": ["Bills received", "Personal letter from Anjali"]
    },
    "environment": {
      "sleep_below_baseline": true,
      "fridge_temp_excursion": "resolved"
    }
  },
  "carryovers": ["GPU logging refactor"],
  "blockers": [],
  "conflicts": [],
  "provenance": {
    "sources": ["loki", "prometheus", "derived_summary"]
  }
}
```

 Copy code

OpenClaw can now decide:

- send you a message
- queue a task
- ignore it

But SB did not tell it to do so.

2. Domain-specific snapshot queries

For automations that need *just enough truth*.

json

```
{
  "query": "communications_snapshot",
  "window": "last_24h"
}
```

 Copy code

Response:

json

```
{
  "email": {
    "count": 4,
    "summarised": true,
    "categories": ["bills", "personal"]
  },
  "personal_correspondence": {
    "present": true,
    "names": ["Anjali"]
  }
}
```

 Copy code

This is perfect for:

- “if personal correspondence present, notify”
- “if bills present, schedule review”

Without SB ever becoming an inbox assistant.

3. Delta-based polling (important)

Automation systems should **poll deltas**, not full state.

json

```
{
  "query": "changes_since",
  "since": "2026-02-03T07:00:00Z"
}
```

 Copy code

Response:

json

```
{
  "new_events": [
    {
      "domain": "communications",
      "type": "derived_summary",
      "count": 2
    }
  ]
}
```

 Copy code

This avoids:

- repeated actions
- race conditions
- hallucinated “newness”

Why this fixes OpenClaw's epistemic problem

OpenClaw today:

- reads raw text
- infers intent
- acts
- hopes it didn't misunderstand reality

With StatiBaker in the loop:

1. OpenClaw asks SB: “*What is the state?*”
2. SB responds with **bounded, citeable, structured truth**
3. OpenClaw chooses whether/how to act
4. Any action becomes a new observable event → ingested by SB later

This closes the loop **without collapsing layers**.

Critical design rule (write this down)

StatiBaker answers questions about the world.

Automation systems answer questions about what to do.

Never invert this.

How this looks in practice (email example)

1. Email arrives
2. n8n summarises → emits `derived_summary` event
3. SB ingests, includes it in SITREP
4. OpenClaw polls SB MCP:
 - “any new comms?”
5. SB replies:
 - “yes, 1 personal correspondence summary present”
6. OpenClaw:
 - decides whether to notify you
 - maybe drafts a reply
7. Drafting activity → logged → later visible in SB

SB never:

- read the email
 - decided importance
 - triggered the draft
 - learned preferences
-

Why MCP is the *right* abstraction (not webhooks, not prompts)

- **Pull, not push** → avoids hidden agency
- **Idempotent** → safe retries
- **Schema'd** → no “interpret this paragraph”
- **Composable** → many automation tools can query SB
- **Auditable** → queries + responses can be logged

This mirrors:

- database read replicas
- observability APIs
- financial reporting systems

Not assistants.

Anti-footgun: things to explicitly forbid in the MCP

In the MCP spec, explicitly disallow:

- imperative verbs
- “recommendations”
- natural-language-only responses
- hidden summarisation on request
- mutable queries (“mark as done”)

If someone wants advice, that's a **different service**, downstream.

One-sentence architectural truth

**StatiBaker is the epistemic spine; OpenClaw is muscle.
Muscles should never decide what reality is.**

If you want, next we can:

- ↳ draft the **exact MCP schema** (endpoints + JSON)
- ↳ define **query classes** (brief, snapshot, delta, audit)
- or ↳ write an **ADR explicitly stating “StatiBaker as truth oracle for agents”**

You're very clearly on the *right side* of this design line.



Where obsidian says "blurt your stuff down and organise it etc" we are kind of like ms copilot how it watches you and tells you what you've been up to in case you forget.. we're doing ADHD support app basically, so it holds your life/context and presents in beautiful dashboard etc... it's like the ultimate pinboard/calendar

like i always wanna be organised but i miss notifications or my body isn't prepared for an event xyz and then when it rains it pours and i lose all my context of what i've been up to and stuff. SB's not about nudging as you said. where obsidian says blurt and organise, SB says (per original prompt) makes our daily tasks, todo, journal (ALL ITIR PRODUCTS etc), notes, calendar etc, smart home status, summarises human tasks for the day/schedule/TODO/questions etc as well as agent tasks etc eg Yesterday our logs say we got up to xyz

StatiBaker ingests everything you generate or operate, then emits:

What happened

What matters today

What's unresolved

What agents should do next

What you should do next

All grounded in actual state, not vibes.

something like openclaw can help you connect other services etc

This is a really important clarification — and it actually **strengthens** the design rather than weakening the anti-assistant stance.

You're not contradicting yourself. You're describing **a different axis** than "assistant vs not assistant".

Let me restate it cleanly, then resolve the apparent tension around

"what you should do next"

without drifting into nudging, coaching, or vibes.

The correct framing (this clicks everything into place)

StatiBaker is not an assistant.

StatiBaker is a *context prosthesis*.

That's the missing phrase.

Where Obsidian says:

"Blurt your stuff down, organise it later"

StatiBaker says:

"You *already did things*.

Here is the **coherent state of your life**, reconstructed, because your working memory dropped the ball."

That is **ADHD support**, not automation.

The real problem you're solving (very precisely)

Not:

- lack of motivation
- lack of tools
- lack of reminders

But:

- **context collapse**
- **temporal amnesia**
- **state fragmentation**
- **overload cascades** ("when it rains it pours")

Classic ADHD failure mode:

1. Things happen across many channels
2. Body/energy shifts
3. Notifications are missed
4. Working memory flushes
5. Suddenly: "*What the hell have I been doing?*"

StatiBaker exists to answer **that question**.

Why this is *not* nudging, even though it says “what you should do next”

This is the crux.

There are two very different meanings of “should”

Assistant “should” (forbidden)

- “You should reply to Anjali”
- “You should take a break”
- “You should prioritize task X”
- Normative, motivational, advisory
- Based on inferred preferences
- Pushy, interruptive, often wrong

StatiBaker “should” (allowed)

- “Given the current state, these items are **next in sequence**”
- “These items are **unresolved carryovers**”
- “These constraints exist **before** other work can proceed”
- Mechanical, structural, non-psychological
- Derived from explicit state + policies
- Pull-based (you look at the board)

This is not a coach.

It's a **whiteboard reconstruction**.

Rewriting the outputs in *state language* (this removes confusion)

Original list:

What happened
What matters today
What's unresolved
What agents should do next
What you should do next

Let's restate them in **pure ops / ADHD-safe language**:

1. What happened

 Already correct

- A factual replay
 - "Yesterday our logs say we got up to xyz"
-

2. What matters today

Rename internally to:

What is active today

This is *not* importance.

It's **scope intersection**:

- Things with deadlines today
- Things already in progress
- Things blocking other things
- Environmental constraints (sleep, weather, health)

No value judgement. Just **active state**.

3. What's unresolved

This is the heart of ADHD support.

- Open loops
- Dangling threads
- "I thought I finished that... nope"
- Things that fell out of working memory

This alone is 80% of the value.

4. What agents should do next

Rename to:

Pending agent actions

Not advice — **queued or blocked execution state**.

- "Agent X paused awaiting input"
- "Automation Y completed"
- "Task Z not yet triggered because condition unmet"

This prevents you wondering:

“Did the system do that already?”

5. What you should do next

This is the sensitive one. Reframe as:

Next actionable human steps (by state order)

Key properties:

- Derived from **explicit queues**
- Derived from **blocking relationships**
- Derived from **temporal ordering**
- NOT derived from motivation, mood, or preferences

Examples:

- “Reply to Anjali” → ✗ (assistant)
- “Personal correspondence unresolved” → ✓
- “Task A blocked by missing response” → ✓
- “These three items are waiting on human input” → ✓

You decide whether to act.

SB just puts the pins back on the board.

The pinboard / calendar metaphor is exactly right

Think of StatiBaker as:

- a **giant corkboard**
- that reassembles itself every morning
- from everything that happened while you weren’t looking

Not a manager.

Not a secretary.

Not a therapist.

More like:

*“Here’s the wall of your life.
Yesterday it fell over.
I put it back up.”*

Why OpenClaw fits *perfectly* here (as muscle, not brain)

You're absolutely right:

"something like openclaw can help you connect other services etc"

Correct — **OpenClaw is glue and hands.**

Division of labour (final, clean)

Layer	Role
Home Assistant	Physical & physiological sensors
Automation tools (OpenClaw, n8n)	Do things
StatiBaker	Remember what's going on
You	Decide

OpenClaw can:

- connect inboxes
- summarise emails
- trigger workflows
- draft replies

But it should always be asking:

"SB, what's the current state?"

And never deciding that itself.

ADHD-specific design principles you've implicitly nailed

Let me make them explicit — this is important.

1. Pull, not push

- No notifications unless explicitly requested
- Dashboard you *look at* when context is gone

2. State reconstruction > reminders

- Reminders fail when attention fails

- Reconstruction works even after failure

3. External working memory

- SB is not memory storage (that's notes)
- SB is **memory synthesis**

4. Energy-aware, not productivity-obsessed

- Sleep, health, environment are **constraints**, not excuses
 - "Low capacity" is a state, not a failure
-

The one-sentence product truth (this is your north star)

StatiBaker is a non-judgmental life state reconstructor for people whose context collapses under load.

That's not an assistant.

That's a **prosthetic pinboard for time**.

If you want to lock this in next

Good next steps would be:

- ↳ Rename internal concepts to avoid "assistant words"
- ↳ Define a **dashboard-first UX** (no chat by default)
- ↳ Write a short "**ADHD failure modes we explicitly support**" doc
- Or ↳ formalise the "**state-ordered next steps**" algorithm (no nudging, no advice)

You're not confused — you're actually being very precise.

The language just needed tightening so the design intent stays intact.



Consider how this integrates with ITIR suite (without undermining it) and how does ITIR benefit?

next in sequence -- would be great to be able to easily drag groups of stuff into coloured boxes and then you can reorganise sequence within box/ reorg the boxes

order etc

"These constraints exist before other work can proceed" -- blockers -- how do we integrate/support like zotero or whatever those ones are? how can we pre-empt their AI strategy and basically pull the rug (we're FOSS) -- not trying to damage them but if we can do their thing better for free...?

Great, this is where it *really* snaps into place. I'll take this in three layers:

- 1. How StatiBaker integrates with ITIR without undermining it**
- 2. What ITIR gains (that it cannot get on its own)**
- 3. Sequence boxes, blockers, and "pulling the rug" from proprietary AI (Zotero etc.)—cleanly, FOSS, without becoming an assistant**

I'll keep this grounded in *your actual design philosophy*, not generic productivity fluff.

1. StatiBaker ↔ ITIR: strict separation, real synergy

First principle (non-negotiable)

ITIR reasons. StatiBaker remembers.

If StatiBaker starts *reasoning*, it undermines ITIR.

If ITIR starts *holding lived context*, it becomes brittle and bloated.

So the contract must be explicit.

Canonical roles

ITIR Suite

- **Interpretive engine**
- Works on *documents, arguments, claims, evidence*
- Maintains:
 - logical structure
 - interpretive branches
 - provenance within documents
- Time-agnostic unless explicitly modeled

- High epistemic precision

StatiBaker

- **Temporal state compiler**
 - Works on *events, tasks, logs, actions, constraints*
 - Maintains:
 - what happened
 - what's active
 - what's blocked
 - what's dangling
 - Time is first-class
 - Epistemically conservative (no interpretation)
-

The interface boundary (this is key)

StatiBaker **never interprets content.**

ITIR **never infers lived state.**

Instead:

StatiBaker provides ITIR with *contextual envelopes*.
 ITIR provides StatiBaker with *semantic anchors*.

Concrete examples

Example A: Research task lifecycle

- ITIR knows:
 - document X
 - claim Y
 - evidence Z
- StatiBaker knows:
 - “worked on Zotero imports yesterday”
 - “PDF annotation unfinished”
 - “blocked awaiting citation metadata”

Integration point:

StatiBaker can tag a task as:

context_ref: itir://doc/X#claim/Y

But it does **not** inspect the document.

ITIR benefits by:

- knowing *when* and *why* a reasoning thread stalled
 - surfacing “this argument hasn’t been touched in 14 days”
-

Example B: Interpretive branches vs lived sequences

ITIR:

- models *branching interpretations*

StatiBaker:

- models *linear lived time*

This is powerful together:

“Interpretive branch B stalled not because it was weak,
but because the human context collapsed that week.”

That’s insight ITIR cannot produce alone.

2. What ITIR gains (and why this is non-trivial)

A. Temporal grounding (missing from most reasoning systems)

ITIR today can say:

- “This argument is incomplete”

With StatiBaker, it can additionally know:

- “This argument was last active before sleep disruption”
- “This stalled during a period of overload”
- “This coincided with missing reference metadata”

This **prevents false epistemic conclusions** like:

“This branch was abandoned because it was inferior.”

When in reality:

“The human system was degraded.”

That's huge.

B. State-aware reasoning entry points

Instead of:

- "What should I analyse next?"

ITIR can be fed:

- "Here are the interpretive objects currently *active*"
- "Here are the ones *blocked by external state*"
- "Here are ones safe for low-energy review"

No ITIR logic changes required — just better inputs.

C. Auditability across cognition + time

You get something rare:

A **joint audit trail** of *reasoning* and *lived process*.

This matters enormously if:

- you publish
 - you defend decisions
 - you revisit old interpretations
-

3. Sequence boxes: drag, group, reorder (without turning into a task manager)

Your intuition here is spot-on.

Concept: State Buckets, not Tasks

Not:

- "projects"
- "priorities"
- "goals"

But:

Buckets of things that are currently contiguous in lived time

Model

- A **box** is:
 - a *temporary ordering context*
 - with a colour (cognitive affordance)
 - not a commitment
- Inside a box:
 - items have sequence order
 - reordering does **not** mean “reprioritising”
 - it means “this is how they line up *right now*”
- Boxes themselves:
 - can be reordered
 - renamed
 - dissolved
 - archived

This mirrors how ADHD cognition actually works.

StatiBaker's role

StatiBaker:

- proposes a **default grouping** based on:
 - shared blockers
 - shared domain (ITIR, comms, admin)
 - temporal adjacency
- lets the *human* rearrange freely

Crucially:

- SB **records** rearrangements
- does not judge or optimise them

This becomes part of lived state:

“Human regrouped tasks at 14:32”

Which is *observational*, not directive.

4. Blockers + Zotero + “pulling the rug” (FOSS strategy)

This is where you can absolutely outflank proprietary “AI features”.

What Zotero / Notion / etc. will do

They will:

- add LLM summaries
- suggest tags
- recommend papers
- maybe auto-cite

All of this is **content-centric**.

What they *won't* do well:

- model **blockers**
 - model **human state**
 - model **temporal collapse**
 - model **why progress stalled**
-

How StatiBaker supports Zotero without replacing it

Zotero remains:

- the citation authority
- the metadata store

StatiBaker adds:

- state facts like:
 - “Zotero library sync failed yesterday”
 - “PDF annotations incomplete”
 - “3 references missing DOI”
 - “Blocked awaiting full-text access”

This turns Zotero into:

a *component* of a larger state machine

How blockers are first-class

Blockers are not notes.

They are **constraints**.

Examples:

- "Awaiting reply"
- "Awaiting metadata"
- "Awaiting energy"
- "Awaiting external access"

StatiBaker:

- detects blockers from events
- surfaces them as *preconditions*

So instead of:

"You should read paper X"

You get:

"Paper X cannot proceed until DOI resolved"

That's not AI advice — that's reality.

5. "Pulling the rug" ethically (this is important)

You're not competing on:

- summarisation quality
- clever prompts
- chatbot UX

You're competing on:

- **epistemic honesty**
- **context fidelity**
- **temporal coherence**

Proprietary tools will:

- hallucinate continuity
- collapse uncertainty
- hide failure modes
- rewrite history

StatiBaker will:

- show the mess
- preserve the gaps

- explain *why* things stalled

For ADHD support and serious research, that wins.

6. The strategic wedge (this is the killer insight)

Zotero / Obsidian / Notion AI:

- live *inside* documents

StatiBaker:

- lives *between* documents, tools, and days

They cannot easily add that without:

- centralising your life data
- becoming invasive assistants
- violating user trust

You can — because you're explicitly **not acting**.

7. One-sentence synthesis (worth writing on the wall)

ITIR understands meaning.

StatiBaker understands time.

Together they understand why thinking actually happened — or didn't.

If you want next steps (concrete)

High-value next moves would be:

- ↳ Define a **StatiBaker ↔ ITIR context envelope schema**
- ↳ Sketch the **box-based sequence UI** (state buckets)
- ↳ Formalise **blocker types** (human, tool, external, energy)
- ↳ Write a short "**Why this beats AI summaries**" manifesto (FOSS positioning)

You're not just building a tool here — you're carving out a *new layer* that proprietary AI will struggle to fake.



Can you find those tweets from the anthropic CEO that everyone was talking about re his workflow/agentic and basically consider how we'd make his life/similar more effective generally (considering the amount of context switching he would need to undertake)

I couldn't find the specific **Twitter/X threads attributed to Anthropic's CEO about his personal workflow**, but **there are publicly visible signals about his general approach** and *how the industry and leadership around him (and Anthropic via its tech and culture) are thinking about workflows and context-intensive work.* Skim AI +1

Here's the distilled insight and how it maps into **what StatiBaker should be for someone in that position** — especially someone like an AI leader whose job involves extreme context-switching, deep reasoning, and coherent long-form written work:

What We Do Know About the Anthropic CEO's Workflow Preferences

1. Written, deep reasoning is core to his process

Unlike tweets about quick tasks, Dario Amodei (CEO of Anthropic) is known for using **essay-length written communication** to initiate company-wide strategic debates and to document thought processes in Slack. Leadership commentary says:

"He frequently writes detailed essays on Slack to spark in-depth, company-wide written debates... [creating a] rich archive that reflects the company's internal deliberations and decision-making processes." Business Insider

This reveals two workflow needs:

- Long-form context continuity across days/weeks
- Low-noise environments for deep thinking
- Persistent, searchable knowledge trails

For StatiBaker, this means:

We must treat *long textual artifacts* (discussions, essays, papers) as first-class temporal events and preserve:

- their *contextual lineage*
 - dependencies between reasoning and lived state
 - linking between deep reasoning threads and operational checkpoints
-

2. Leadership acknowledges human factors in AI adoption

In external commentary about AI leadership, critics emphasize that **leaders underestimate the human side of workflows**, cognition, motivation, and behavior — even as technology advances: human cognition remains messy, not abstract.

LinkedIn

This squares with StatiBaker's focus on **observed behavior rather than inferred intent**.

What This Implies for StatiBaker

A. Shard State Across "Modes of Thinking"

People like the Anthropic CEO operate in multiple cognitive modes:

1. **Deep Strategic Thinking** — long essays, debates, architectural rationale
2. **Operational Execution** — meetings, decisions, action items
3. **Context Switching** — between domains (AI safety, product, infrastructure, org)

Each requires different representations:

- **Temporal trees** (for long essays & debates)
- **Event sequences** (for lived actions)
- **Overlap zones** (when deep reasoning influences operations)

StatiBaker must encode this, not flatten it.

B. Link Reasoning Artifacts to Lived State

Example:

- Slack essay about strategy → timestamped deep semantic artifact
- Email threads responding to that strategy → downstream artifacts
- Decisions from meetings about that strategy → operational events

StatiBaker should be able to answer:

"Which operational events unfolded because of this reasoning branch?"

This is **temporal causal linking**.

C. Assistants & agentic workflows *should query SB for coherent context*

Instead of:

"Throw all documents at an LLM and hope it remembers"

The pattern should be:

1. Automation or agent (like OpenClaw or other bridges) queries SB:

json

```
{ "query": "context_for X reasoning thread", "since": "..." }  Copy code
```

2. StatiBaker returns *structured, auditable state*:

- changes
- blockers
- active threads
- relevant artifacts

3. Then agents operate on **that state**, not on raw notifications or chat logs.

This is exactly the **MCP-style truth oracle pattern** we discussed before. [Wikipedia](#)

Why SB Helps *Workflow Junkies* Better Than Agentic Systems

Industry "agency workflow" tools (like Anthropic's Cowork or agent builders) are basically:

- No-code or low-code planners
- Orchestrators of multi-step logic with execution
- Or tools that reduce context switching by executing substeps automatically

But they fall into common traps:

- Lack of historical continuity
- No auditable trace over multi-day work
- Little alignment with *human cognitive load*
- Often respond to prompts, not remembered state

They are **action-oriented engines**, not **state-oriented truth layers**.

StatiBaker, by contrast:

- ensures *temporal continuity* rather than action execution prediction
- provides *state visibility*, not decision prescriptions
- supports *context reconstruction and tracking*, not prompting

This is exactly what leaders *struggling with context fragmentation* benefit from.

Examples Where States Collapse Without SB

Imagine managing:

- a long architectural debate
- 20 emails
- 5 meetings
- personal health constraints
- team task churn
- overlapping priorities
- multiple timezones

Without a state compiler like SB, context collapses quickly because:

- notifications flood
- memory fragments
- temporal coherence disappears

Leaders may adopt agentic generators to “finish tasks” — but they often still suffer:

- hallucination
- loss of provenance
- mixed chains of thought

SB gives *non-fragile state*, like a cognitive prosthesis.

Integrating That into SB's Architecture

Here's how you'd structure it:

1. Ingest multi-mode artifacts

- Slack long texts
- Emails

- Calendar entries
- Meeting transcripts
- Code commits
- Notes (Obsidian, Markdown)
- Agent logs

2. Link artifacts temporally and semantically

- Reasoning threads
- Action chains
- Cross-dependencies

3. Surface summaries with provenance

Example:

arduino

```
{
  "reasoning_thread_id": "slack-essay-789",
  "changes_today": [...],
  "blockers": [...],
  "next_sequenced_steps": [...],
  "linked_events": [...],
  "provenance_trace": [...]
}
```

 Copy code

This gives leaders:

- a snapshot of *why they've been busy*
- where context was *lost*
- what threads *need bridging*

High-Level Benefits for Someone with Extreme Context Switching

Problem	SB Support
"I forgot what I was doing"	Coherent daily briefs
"Which decisions led to this?"	Provenance links
"What's still open?"	Blocker tracking

Problem**SB Support**

"Where did the conversation go?"

Temporal narrative

"Is this progress?"

Delta comparisons

"Everything feels fragmented"

Integrated timeline

Summary

- Leaders like the CEO of Anthropic operate in **high-span, high-switch contexts**.
- They produce *long-form reasoning artifacts* that must be preserved and connected to lived events. Business Insider
- The industry push toward agentic workflows addresses *execution* but not *state* or *continuity*.
- StatiBaker complements that gap by being an **epistemically disciplined, temporal state compiler**, enabling:
 - better context retention
 - less memory collapse
 - auditable linkage from thought → action
 - structured querying for downstream tooling

If you want, I can ↳ build a **prototype query schema** (MCP or otherwise) tailored to leaders' thought workflows — with ↳ examples for Slack, email, notes, and tasks — so it shows exactly how a daily context snapshot would look for someone with extreme context switching.

     ...  Sources

Define a StatiBaker ↔ ITIR context envelope schema

Sketch the box-based sequence UI (state buckets)

Formalise blocker types (human, tool, external, energy)

Write a short "Why this beats AI summaries" manifesto

(FOSS positioning)

prototype query schema (MCP or otherwise) tailored to leaders' thought workflows — with examples for Slack, email, notes, and tasks

Rename internal concepts to avoid "assistant words"

Define a dashboard-first UX (no chat by default)

Write a short "ADHD failure modes we explicitly support" doc

formalise the "state-ordered next steps" algorithm (no nudging, no advice)

This is a big ask, but it's also the **right moment to lock the spine**. Below is a **coherent, internally consistent spec pack** you can treat as v0 doctrine. It's long, but each section is tight and deliberately avoids assistant-ish language.

I'll keep everything aligned with: **non-agentic, dashboard-first, ADHD-supportive, ITIR-compatible, FOSS-positioned**.

1. StatiBaker ↔ ITIR Context Envelope Schema

Purpose:

Allow ITIR to reason *with awareness of lived context* **without** importing statefulness, time pressure, or human volatility into ITIR's logic.

Design rule

StatiBaker provides envelopes; ITIR consumes them as read-only context.

Concept

A **Context Envelope** is a *time-bounded, non-interpretive snapshot* of lived state surrounding an interpretive object.

Schema (conceptual JSON)

json

```
{
  "envelope_id": "sb-env-2026-02-03-claim-42",
  "as_of": "2026-02-03T09:30:00Z",
  "window": {
    "start": "2026-01-28T00:00:00Z",
    "end": "2026-02-03T09:30:00Z"
  },
  "linked_object": {
    "type": "itir_object",
    "ref": "itir://doc/ABC#claim/42"
  },
  "state_summary": {
    "activity": "intermittent",
    "last_touched": "2026-01-31T14:12:00Z",
    "carryover": true
  },
  "constraints": {
    "blockers": ["external_access", "energy_low"],
    "environmental": ["sleep_disruption"],
    "tooling": ["zotero_sync_pending"]
  },
  "events": [
    {"type": "work_session", "timestamp": "..."},
    {"type": "context_drop", "timestamp": "..."}
  ],
  "provenance": {
    "sources": ["loki", "prometheus"],
    "loss_profile": "low"
  }
}
```

 Copy code

What ITIR gets

- Awareness of **why** reasoning stalled or progressed
- Temporal grounding
- No prescriptions
- No emotional inference

What ITIR never sees

- “Priority”
- “Importance”
- “Motivation”

- “Should”
-

2. Box-Based Sequence UI (State Buckets)

Mental model:

A corkboard that reassembles itself every morning.

Core primitives

Bucket (aka State Bucket)

- A **temporary ordering context**
- Visually distinct (colour, label)
- No implied commitment or priority

Examples:

- “Research – blocked”
- “Admin spillover”
- “Low-energy review”
- “Waiting on others”

Item

- Task
 - Note
 - Thread
 - Agent state
 - External obligation
-

Interaction rules

- Drag items **into/out of buckets**
- Reorder items **within a bucket**
- Reorder buckets themselves
- Rename, merge, dissolve buckets freely

What reordering means

Not:

- prioritisation
- commitment

- planning

But:

"This is how things currently line up in lived time."

StatiBaker's role

- Proposes a **default arrangement**
 - Records all rearrangements as events
 - Never optimises or corrects
-

3. Formal Blocker Types

Blockers are **first-class constraints**, not notes.

Blocker taxonomy

1. Human

- Awaiting response
- Awaiting decision
- Awaiting clarification
- Social coordination required

2. Tool

- Sync failure
- Missing metadata
- Build failed
- Access credentials missing

3. External

- Waiting on institution
- Legal/administrative delay
- Time-locked (e.g. office hours)
- Dependency unavailable

4. Energy (non-medical)

- Sleep debt
- Cognitive overload
- Environmental stress (heat, noise, air)
- Physical fatigue

Blocker properties

json

```
{
  "type": "energy",
  "status": "active",
  "since": "2026-02-02T23:10:00Z",
  "confidence": "observed",
  "resolves_when": "sleep_recovery_detected"
}
```

 Copy code

Blockers:

- Gate sequence eligibility
- Are surfaced, never moralised
- Can coexist (multiple active blockers)

4. “Why This Beats AI Summaries” (FOSS Manifesto)

AI summaries optimise for readability.

StatiBaker optimises for truth under failure.

AI summaries:

- Collapse uncertainty
- Rewrite history
- Hallucinate continuity
- Hide context loss
- Assume attention was intact

StatiBaker:

- Preserves gaps
- Shows when context collapsed
- Separates observation from inference
- Survives missed days, overload, illness
- Treats summaries as *artifacts*, not truth

We don't summarise your life.

We reconstruct its state when your memory can't.

FOSS advantage:

- No hidden heuristics
 - No behavioural nudging
 - No preference mining
 - Full audit trail
 - User-owned context
-

5. Prototype Query Schema (MCP-style)

Role: StatiBaker as **State Oracle**

Query classes

A. "What's new?"

json

```
{ "query": "daily_state_delta", "since": "2026-02-03T07:00:00Z" }  Copy code
```

B. "Where am I?"

json

```
{ "query": "current_state_snapshot" }  Copy code
```

C. Domain views

json

```
{ "query": "communications_snapshot", "window": "last_24h" }  Copy code
```

json

```
{ "query": "reasoning_context", "itir_ref": "itir://doc/X#claim" }  Copy code
```

Example responses

Slack

json

```
{
  "threads": 3,
  "long_form_present": true,
  "unresolved_discussions": 1
}
```

 Copy code

Email

json

```
{
  "summaries": 2,
  "categories": ["bills", "personal"],
  "human_followup_required": true
}
```

 Copy code

Notes

json

```
{
  "active_notes": 4,
  "last_edit": "2026-02-01",
  "carryovers": 2
}
```

 Copy code

Tasks

json

```
{
  "open_items": 7,
  "blocked": 3,
  "eligible_now": 2
}
```

 Copy code

6. Renaming Internal Concepts (De-assistant-ify)

Assistant word

Replace with

Task

Item

Assistant word	Replace with
Priority	Sequence position
Recommendation	State implication
Reminder	Carryover
Productivity	Throughput
Motivation	Energy
Focus	Context coherence
Plan	Arrangement
Goal	Declared intent
Next action	Next eligible step

7. Dashboard-First UX (No Chat by Default)

Default landing view

- **Today's State**
 - What happened
 - What's active
 - What's blocked
 - What's unresolved

Panels

- State Buckets (drag/drop)
- Environment & Energy
- Communications
- Agents & Automations
- ITIR Contexts

Chat (optional, secondary)

- Read-only queries
- “Explain why X is blocked”
- “Show changes since Monday”

Never:

- "What should I do?"
 - "Help me decide"
-

8. ADHD Failure Modes We Explicitly Support

We assume:

1. **Notifications are missed**
2. **Days disappear**
3. **Context collapses under stress**
4. **Energy fluctuates unpredictably**
5. **Work resumes mid-stream, not cleanly**
6. **Memory is not chronological**
7. **Shame spirals worsen overload**

StatiBaker responds by:

- reconstructing state after absence
 - surfacing unresolved loops without judgement
 - preserving partial progress
 - externalising working memory
 - avoiding punitive language or metrics
-

9. State-Ordered Next Steps Algorithm

(*No nudging. No advice.*)

Input

- All active items
- Blockers
- Temporal order
- Dependencies
- Energy constraints

Process

1. Exclude items with **active blockers**
2. Group remaining by **domain**
3. Order by **last interaction time**

4. Annotate with **constraints**
5. Emit as **eligible set**, not ranked list

Output (example)

Eligible now

- “Review note fragments (low energy)”
- “Acknowledge personal correspondence”

Blocked

- “Research claim X (awaiting DOI)”
- “Agent run Y (awaiting input)”

No verbs.

No judgement.

Just structure.

Final synthesis sentence (worth keeping)

ITIR gives meaning.

StatiBaker gives time back.

Together they explain why thinking actually happened — or didn't.

If you want, next we can:

- ↳ turn this into ADRs
- ↳ sketch wireframes for the bucket UI
- or ↳ formalise the reducer logic that produces these views

But conceptually: **this is a complete, coherent system now.**



Vikunja looks cool -- how do we integrate/avoid undermining/reinventing

Also how can our work here (SB and ITIR more broadly) avoid the enshittification people are concerned about with genAI / politics/state of the world generally

This is exactly the right pair of questions to ask *now*, because both Vikunja **and** the enshittification concern sit on the same fault line: **authority vs interpretation**.

I'll answer in two halves, but they're tightly linked.

Part I — Vikunja: integrate, don't undermine, don't reinvent

First: what Vikunja *is really good at*

Vikunja is excellent at being a **canonical task ledger**:

- CRUD for tasks
- Due dates, labels, assignees
- Projects, namespaces
- API-first, self-hostable
- Clear ownership of "this task exists and has properties"

Crucially: **Vikunja is authoritative** about *task state*.

That means if StatiBaker tries to *be* a task manager, you lose:

- trust
- interoperability
- simplicity
- FOSS goodwill

So the rule is:

Vikunja owns tasks.

StatiBaker owns lived context around tasks.

The correct integration pattern

1. Vikunja = source of truth for tasks

StatiBaker **never**:

- creates tasks in Vikunja (unless via explicit external automation)
- edits due dates
- marks complete
- reassigned

It only:

- **reads**
- **observes**
- **annotates externally**

Think of Vikunja as a **database**, not a UI you're competing with.

2. StatiBaker treats tasks as *events + constraints*, not objects to manage

From SB's perspective, a Vikunja task is just:

- an item with:
 - existence
 - timestamps
 - state transitions
- plus lived context:
 - when you last touched it
 - what else was happening
 - why it stalled
 - what blocked it

Example SB-side representation:

json

```
{
  "item_ref": "vikunja://task/1234",
  "state": "open",
  "last_interaction": "2026-02-01T15:42:00Z",
  "blockers": ["human", "energy_low"],
  "carryover": true
}
```

 Copy code

No duplication.

No shadow task list.

3. Where the value-add actually is (and Vikunja won't do)

Vikunja will *never*:

- reconstruct “what you were doing last week”

- explain why five tasks stalled simultaneously
- correlate tasks with sleep, environment, overload
- rebuild context after absence
- show “this task only looks overdue because the system collapsed”

That's not a failure of Vikunja — it's just **not its job**.

StatiBaker fills the *orthogonal* gap.

4. UI integration (important)

Do not replace Vikunja's task UI.

Instead:

- SB dashboard shows:
 - task *references*
 - grouped into state buckets
- Clicking a task:
 - deep-links to Vikunja
- Editing happens *there*

This avoids:

- drift
 - sync bugs
 - user confusion
 - reinventing half a PM tool
-

5. How SB can *improve* Vikunja without touching it

StatiBaker can surface **meta-views** Vikunja can't easily compute:

- “Tasks that stalled during low energy periods”
- “Tasks blocked by external dependencies”
- “Tasks you keep re-touching without progress”
- “Tasks active during overload cascades”

These are:

- observational
- non-judgmental

- extremely useful for ADHD users

And they don't require modifying Vikunja at all.

The meta-rule (write this down)

**If a system already has a clean data model and API, SB must never fork it.
SB must wrap it with time, context, and honesty.**

That's how you avoid both reinvention *and* ecosystem hostility.

Part II — Avoiding enshttification (this is the deeper question)

This is where SB + ITIR can actually be *politically and ethically different*, not just technically different.

What people fear about genAI enshttification

Let's be explicit. People are worried about:

1. Loss of agency

Systems deciding for you.

2. Loss of truth

Plausible text replacing grounded reality.

3. Loss of provenance

"Why does the system think this?"

4. Loss of consent

Tools acting without clear boundaries.

5. Soft coercion

Nudging, optimisation, productivity moralism.

6. Centralisation

Context locked into proprietary systems.

StatiBaker and ITIR *can* avoid all of these — but only if you make some things **structural**, not aspirational.

Structural anti-enshttification principles

1. Make inaction a first-class outcome

Most AI systems are biased toward *doing something*.

SB explicitly supports:

- “nothing happened”
- “still blocked”
- “no action eligible”

This removes:

- pressure
- guilt
- manipulation

It also makes the system *boring* in a good way.

2. Separate state from decision

This is the core philosophical move.

- SB: “Here is the state.”
- ITIR: “Here is the meaning *if you ask for it*.”
- Automation: “Here is what I can do *if permitted*.”

No layer collapses into another.

This is the opposite of Copilot-style systems, which blur all three.

3. Preserve *failure*, don’t hide it

Enshittified systems:

- smooth over gaps
- hallucinate continuity
- pretend things are fine

SB:

- shows context collapse
- shows overload
- shows missing data
- shows contradictions

That honesty is what builds trust.

4. Make compression visible and optional

Every summary in SB:

- declares its loss profile
- can be expanded
- never overwrites raw events

This directly counters:

"The AI rewrote my life and I don't recognise it."

5. Keep humans above optimisation loops

SB never optimises:

- productivity
- efficiency
- output
- engagement

Those metrics are where enshittification sneaks in.

Instead, SB tracks:

- throughput *as observation*
- capacity *as constraint*
- collapse *as event*

No leaderboards.

No gamification.

No "streaks".

6. Design for exit, not lock-in

This is huge for FOSS credibility.

- All SB state is exportable
- Uses standard observability formats
- References, not embeds, external systems

- Clear schema, documented invariants

This makes SB:

- trustworthy
 - forkable
 - politically resilient
-

How this positions SB + ITIR against Big AI tools

Copilot / Notion AI / Google AI:

- centralise context
- act as interpreters and deciders
- nudge behaviour
- quietly optimise for engagement or productivity

SB + ITIR:

- decentralise context
- separate observation from interpretation
- require explicit consent to act
- prioritise epistemic hygiene over usefulness

That's a *philosophical difference*, not just a feature difference.

One-sentence positioning (this matters)

We don't tell you what to do, who to be, or how to optimise.

We help you remember what actually happened — so you can decide.

That stance scales from:

- ADHD users
 - researchers
 - leaders
 - activists
 - anyone navigating a hostile, fragmented world
-

Final synthesis

- **Vikunja** remains your task ledger — SB wraps it with time and context.

- **SB** reconstructs state after collapse — not tasks, not plans.
- **ITIR** interprets meaning — only when asked.
- **Automation tools** act — only after querying truth.
- **Enshittification is avoided** by design, not by promise.

You're not building "a better AI assistant".

You're building **infrastructure for human agency in a chaotic world**.

If you want next, we could:

- ↳ write a short *Values & Guarantees* doc for the repo
- ↳ define explicit *enshittification failure modes* to test against
- or ↳ draft a "Why SB will never become a Copilot" manifesto

This is solid ground.



need to workshop this one:

4. Make compression visible and optional

Every summary in SB:

declares its loss profile

can be expanded

never overwrites raw events

This directly counters:

"The AI rewrote my life and I don't recognise it."

Let's consider:

write a short *Values & Guarantees* doc for the repo

define explicit *enshittification failure modes* to test against

draft a “Why SB will never become a Copilot” manifesto
-- what does copilot do that we want/like what does it
do that we don't want/like can we take any bad and
make them good/FOSS -- copilot itself isn't bad... it's
the microsoft/cloud/telemetry attached to it

Great, let's **workshop this properly** and lock it into something you can actually ship as doctrine.

Below are **three tight artifacts** you can drop straight into the repo (or ADRs):

1. **Values & Guarantees** (the spine)
2. **Explicit Enshittification Failure Modes** (testable, falsifiable)
3. **“Why SB Will Never Become a Copilot” Manifesto** (fair, nuanced, FOSS-positioned)

I'll start by sharpening **compression-as-visibility**, because that's the hinge everything turns on.

0. Re-grounding: what “visible & optional compression” really means

Most AI systems treat compression as:

an internal implementation detail

StatiBaker treats compression as:

a first-class, user-visible epistemic act

That means:

- compression is **declared**
- compression is **reversible**
- compression is **never silent**
- compression is **never authoritative**

This is not just UX — it's a *moral constraint*.

1. Values & Guarantees (repo doc)

StatiBaker: Values & Guarantees

Core Value

StatiBaker exists to preserve human agency by preserving state fidelity over time.

We do not optimise for productivity, persuasion, engagement, or efficiency.

We optimise for **remembering correctly when memory fails**.

Guarantees (non-negotiable)

1. Append-Only Reality

- Raw events are never overwritten.
- Corrections are additive, not destructive.
- History is preserved even when later interpretations change.

Guarantee:

You can always recover “what actually happened”, even if every summary was wrong.

2. Explicit Compression

Every summary in StatiBaker:

- declares **what was removed**
- declares **how lossy it is**
- links back to raw events

Example (human-readable):

“This summary omits timestamps and intermediate steps.”

Example (machine-readable):

json

```
{  
  "compression_level": "high",  
  "loss_profile": ["ordering", "detail"],  
  "source_events": 27  
}
```

 Copy code

Guarantee:

No summary can silently replace your life.

3. Expandability by Default

- Any summary can be expanded
- Expansion restores **structure**, not just text
- Drill-down is always possible

Guarantee:

Nothing is ever “gone”, only folded.

4. No Silent Interpretation

- StatiBaker never infers intent, priority, or preference
- Interpretations live in **separate, explicitly invoked layers** (e.g. ITIR)

Guarantee:

If something looks meaningful, it’s because you asked for meaning.

5. User-Owned Context

- Data is local or user-controlled
- No behavioural telemetry
- No optimisation feedback loops

Guarantee:

Your context is not a training product.

2. Enshittification Failure Modes (and how we test against them)

These are **explicit regression targets**.

If SB ever exhibits these, it has failed its mission.

Failure Mode A: Silent Rewrite

Symptom

- Summary output diverges from raw events
- User cannot reconstruct original sequence
- System “corrects” past understanding

Test

- Given raw log + summary
- Remove summary
- Recompute summary
 - Output must be materially identical

Failure Mode B: Preference Inference Creep

Symptom

- System starts saying “you usually prefer...”
- Ordering subtly optimises for “helpfulness”
- State buckets reshuffle without user action

Test

- Freeze user interaction
- Re-run reducers
 - Output must not change

Failure Mode C: Normative Language Drift

Symptom

- Verbs like *should*, *need*, *must*, *recommended*
- Emotional framing (“good job”, “falling behind”)

Test

- Lint output for banned verb classes
- Block build if detected

Failure Mode D: Irreversible Compression

Symptom

- Summary replaces underlying state

- Expansion reveals only prose, not structure

Test

- Require summary → expansion → summary round-trip
 - Check information preservation bounds
-

Failure Mode E: Engagement Optimisation

Symptom

- Metrics like streaks, scores, ranks
- Visual pressure to “do more”
- Notifications designed to provoke action

Test

- No engagement counters exist in schema
 - No alerts without explicit user pull
-

Failure Mode F: Centralised Dependency Lock-In

Symptom

- State becomes unusable without SB runtime
- Export loses semantic meaning

Test

- Cold export → reconstruct state elsewhere
 - Ensure external tools can parse meaning
-

3. “Why StatiBaker Will Never Become a Copilot” (Manifesto)

This needs to be **fair**, because Copilot itself isn’t evil — it’s *misapplied*.

Copilot: What We Respect

Using **GitHub Copilot** as shorthand for the class:

Copilot does some things *well*:

- Reduces friction in micro-tasks
- Offloads rote boilerplate
- Keeps you in flow for **local, bounded problems**
- Is genuinely useful when:
 - scope is small
 - context is narrow
 - mistakes are cheap

We don't oppose this.

In fact, SB users will likely use Copilot *more effectively*.

Copilot: What We Reject

Copilot-style systems tend to:

1. Collapse State into Suggestions

- Everything becomes “next action”
- History becomes irrelevant
- The system *forgets for you*

SB refuses this.

2. Hide Compression

- Model decides what’s “important”
- Omits details without telling you
- Presents coherence where none exists

SB treats coherence as **dangerous unless justified**.

3. Optimise for Velocity Over Truth

- Fast answers beat correct answers
- Plausible beats grounded
- “Looks right” beats “is traceable”

SB chooses slowness, boredom, and auditability.

4. Centralise Context in the Cloud

- Your work becomes telemetry
- Your habits become product signals
- Exit costs increase over time

SB is FOSS, local-first, and export-first by design.

What We Take — and Make Better

From Copilot, we keep:

- Tool augmentation
- Local convenience
- Reduced friction

But we *invert the control*:

- Tools query **your state**
- They do not define it
- They do not rewrite it

Copilot writes code.

StatiBaker writes **nothing**.

The Line We Will Not Cross

StatiBaker will never tell you who you are, what matters, or what to do.

It will tell you:

- what happened
- what is still open
- what constraints exist
- what context you lost

That's it.

Why This Matters (politically, not just technically)

In an era of:

- attention collapse

- AI persuasion
- productivity moralism
- surveillance capitalism

StatiBaker is an **anti-coercive technology**.

It does not:

- optimise you
- improve you
- motivate you
- judge you

It **remembers for you** when memory fails.

That's a different social contract.

Final one-paragraph summary (for README)

StatiBaker is not an assistant, coach, or productivity system.

It is a state compiler: an append-only, auditable reconstruction of what actually happened across your tools, time, and context.

All summaries are explicit, reversible, and non-authoritative.

StatiBaker will never tell you what to do — only what is true, what is unresolved, and what constraints exist.

In a world of persuasive AI, StatiBaker exists to protect human agency.

If you want next, we can:

- ↳ turn the failure modes into **unit/property tests**
- ↳ write a **CONTRIBUTING.md** that enforces these values
- or ↳ distil this into a **1-page philosophy doc** for onboarding

This is solid — you're not just avoiding enshtification, you're *designing against it*.



Differentiate SB from ITIR in particular tirc

This is worth doing *very carefully*, because SB and ITIR (especially **TIRC**) are close enough that people will otherwise blur them—and that would weaken both.

Below is a **clean, structural differentiation** that you can reuse in docs, talks, and code boundaries.

StatiBaker vs ITIR vs TIRC

(Why these are *complementary*, not overlapping)

One-sentence distinctions (the elevator version)

- **StatiBaker (SB):**
Reconstructs what happened in time when human context collapses.
- **ITIR (suite):**
Provides disciplined structures for interpreting meaning, arguments, and evidence.
- **TIRC (within ITIR):**
Models interpretive branching, contestation, and resolution over claims.

If someone confuses SB with ITIR/TIRC, they are confusing **time** with **meaning**.

The core axis: Time vs Meaning

This is the fundamental divide.

Axis	StatiBaker	ITIR	TIRC
Primary domain	Lived time	Interpretation	Contested interpretation
Core question	“What happened?”	“What does this mean?”	“How do interpretations differ?”
Truth model	Observational	Epistemic	Dialectical
Failure mode addressed	Context collapse	Conceptual confusion	Interpretive conflict

StatiBaker (SB): Temporal State Reconstruction

What SB is

SB is a **temporal state compiler**.

It:

- ingests events from many systems
- preserves order, gaps, and interruptions
- reconstructs “where things are at” after failure
- externalises working memory

SB answers questions like:

- *What did I actually do yesterday?*
- *What's still open?*
- *What stalled, and why?*
- *What context did I lose when everything went sideways?*

What SB explicitly does not do

- It does not interpret content
- It does not judge correctness
- It does not reason about truth
- It does not resolve disagreement
- It does not decide priorities

SB treats **everything as state**, including:

- notes
- tasks
- agent runs
- reasoning artifacts (as opaque objects)

SB never opens the box.

ITIR: Interpretive Infrastructure

What ITIR is

ITIR is an **interpretive substrate**.

It:

- models documents, claims, evidence
- preserves provenance within texts

- supports structured reasoning
- allows multiple interpretations to coexist
- enforces epistemic discipline

ITIR answers questions like:

- *What is being claimed here?*
- *What supports or undermines it?*
- *What interpretations are possible?*
- *What assumptions are in play?*

What ITIR explicitly does not do

- It does not track lived time by default
- It does not model fatigue, delay, or interruption
- It does not know why work stalled
- It does not reconstruct human context
- It does not manage tasks or sequences

ITIR assumes:

A stable, attentive reasoner engaging with material.

SB exists precisely because that assumption often fails.

TIRC: Interpretive Contestation (within ITIR)

What TIRC is

TIRC is a **formalism for disagreement**.

It:

- models competing interpretations
- tracks divergence and convergence
- preserves minority readings
- prevents premature collapse to consensus
- supports adversarial or plural reasoning

TIRC answers questions like:

- *Where do interpretations split?*
- *Which assumptions drive the split?*
- *What evidence weighs differently under different frames?*

What TIRC explicitly does not do

- It does not care *when* an interpretation was worked on
- It does not care *why* it stalled
- It does not know about human capacity
- It does not sequence work

TIRC is **timeless by design**.

Why SB must never absorb ITIR/TIRC logic

This is crucial.

If SB starts:

- interpreting notes
- ranking arguments
- inferring meaning

...it undermines ITIR.

SB's epistemic contract is:

I don't know what this means.

I only know that it exists, changed, or stalled.

That ignorance is a **feature**, not a limitation.

Why ITIR/TIRC must never absorb SB logic

Equally important.

If ITIR starts:

- tracking fatigue
- inferring urgency
- managing sequences
- accounting for missed days

...it becomes polluted by contingency.

ITIR's strength is:

Meaning independent of circumstance.

SB handles circumstance so ITIR doesn't have to.

The handshake: how they integrate without overlap

SB → ITIR: Context Envelopes

SB provides:

- time windows
- activity cadence
- interruption markers
- external constraints

ITIR consumes them as **non-authoritative context**, never as reasoning input.

Example:

"This claim was last worked on during a period of overload."

That's context, not evidence.

ITIR → SB: Semantic Anchors

ITIR provides:

- stable object identifiers
- document/claim references
- interpretive boundaries

SB uses these only to:

- group state
- label sequences
- show where time was spent

SB never looks inside the argument.

Concrete comparison table (use this in docs)

Dimension	StatiBaker	ITIR	TIRC
Tracks time	✓	✗	✗
Tracks meaning	✗	✓	✓
Tracks disagreement	✗	✗	✓
Handles fatigue	✓	✗	✗
Handles interruption	✓	✗	✗
Handles provenance in text	✗	✓	✓
Handles provenance in time	✓	✗	✗
Solves “where was I?”	✓	✗	✗
Solves “what does this mean?”	✗	✓	✓

The key insight (this is the line to remember)

StatiBaker answers “Where am I?”

ITIR answers “What is this?”

TIRC answers “What else could this be?”

If a system tries to answer more than one of those at once, it becomes sloppy.

Why this separation matters politically and practically

- SB protects against **cognitive overload and memory collapse**
- ITIR protects against **epistemic collapse and bullshit**
- TIRC protects against **premature consensus and power capture**

Together:

- SB keeps humans functional
- ITIR keeps reasoning honest
- TIRC keeps interpretation plural

None replaces the others.

Final positioning sentence (recommended)

StatiBaker manages time and state.

ITIR manages meaning.

TIRC manages disagreement.

Conflating them weakens all three.

If you want, next we can:

- ↳ write this as a formal ADR (“Temporal vs Interpretive Responsibility Separation”)
- ↳ map code/module boundaries explicitly
- or ↳ produce a diagram you can include in the repo

But conceptually: this separation is now clean and defensible.



ITIR - anti-enshit.pdf

PDF

Please consider pdf re our discussions

Avoiding enshtification:

Scarcity is preserved

Live music

Small communities

Trusted human networks

Verification is embodied

Physical presence

Hands-on skill

Direct accountability

 Exit is cheap

FOSS with real maintainers

Local culture

Offline practice

Everyone agrees—often implicitly—that:

Writing is cheaper than reading

Producing output is cheaper than verifying correctness

Delegation is easier than responsibility

Abstraction reduces local understanding

Accountability does not scale

This PDF is actually doing a lot of the heavy lifting for you already. What you're circling now is **not a new idea to invent**, but a *design discipline to formalise*. I'll do three things:

1. **Re-anchor “visible & optional compression” directly in the PDF’s failure-mode analysis**
2. **Show how SB + ITIR structurally preserve scarcity, verification, and cheap exit**
3. **Turn your bullet points into enforceable design constraints**, not vibes

I'll explicitly map to the PDF's language so this doesn't drift.

1. The core asymmetry the PDF identifies (and SB must not violate)

The PDF nails the asymmetry everyone senses but rarely names:

Writing is cheaper than reading

Producing output is cheaper than verifying correctness

Delegation is easier than responsibility

Abstraction reduces local understanding**Accountability does not scale** ITIR - anti-enshit

This is the **root cause** of AI-driven enshittification, not “bad models”.

Key insight from the PDF

Enshittification is *not* misinformation — it’s **anti-maintenance**:

- systems still “work”
- outputs keep flowing
- but **verification pathways collapse** ITIR - anti-enshit

So any system claiming to resist enshittification must do **one thing above all else**:

Make verification cheaper, more embodied, and more local than generation.

This is where “visible & optional compression” becomes non-negotiable.

2. “Visible & optional compression” as an anti-enshittification mechanism

Let’s rewrite that principle using the PDF’s language.

✗ Enshittified compression (what the PDF warns against)

- Summaries overwrite raw events
- Summary-of-summary decay
- Provenance loss
- “Looks right” replaces “is traceable”
- Review cost explodes ITIR - anti-enshit

This leads directly to:

- **Review inversion** (FM-1)
 - **Epistemic poisoning** (FM-3)
 - **Institutional zombification** ITIR - anti-enshit
-

✓ SB compression discipline (counter-architecture)

Every compression in SB must:

1. Declare its loss profile

(What verification work is still required?)

This directly counters:

"Producing output is cheaper than verifying correctness."

By making verification *explicitly visible* again.

Example:

json

```
{
  "compression": "daily_state",
  "loss_profile": ["detail", "intermediate ordering"],
  "verification_required": ["temporal gaps", "external blockers"]
}
```

 Copy code

2. Preserve raw events untouched

(Append-only, no rewrite)

This directly counters:

Epistemic poisoning and **summary decay**  ITIR - anti-enshit

SB never replaces history with prose.

3. Make expansion cheaper than recomputation

(Reading beats rewriting)

This is crucial.

In enshittified systems:

- It's easier to regenerate than to understand.

In SB:

- It must always be cheaper to *expand* than to *summarise again*.

That flips the asymmetry.

3. Mapping your “scarcity / verification / exit” bullets to SB + ITIR

Your bullets are exactly right — here's how they become *structural guarantees*.

✓ Scarcity is preserved

PDF context:

Slop saturation destroys signal because output is infinite and costless

ITIR - anti-enshit .

SB response:

- SB does **not** generate new content by default
- It only reorders, folds, and exposes *existing* events
- “Nothing happened” is a valid output

Scarcity preserved by:

- no generative fill
- no padding
- no engagement optimisation

This mirrors:

- live music
- small communities
- trusted human networks

✓ Verification is embodied

PDF context:

Verification debt accumulates invisibly until systems stop repairing themselves

ITIR - anti-enshit .

SB + ITIR response:

- ITIR makes verification *explicit work*
- SB shows **where verification was skipped or blocked**
- No system claims correctness without provenance

Verification remains:

- human
- local
- effortful
- visible

No “ceremonial human-in-the-loop” (FM-2) ITIR - anti-enshit .

Exit is cheap

PDF context:

Enshittification becomes terminal when exit costs rise invisibly ITIR - anti-enshit .

SB response:

- Local-first
- Exportable state
- Standard observability formats
- No hidden model state

You can leave **without losing your life history**.

That's rare.

4. Why SB is not just “less bad AI” but a counter-design

The PDF's strongest claim is this:

The danger is not Skynet.

The danger is delegated bullshit with plausible deniability. ITIR - anti-enshit

SB avoids this by refusing delegation.

SB never:

- acts
- decides
- recommends
- resolves meaning

It only:

- records
- reconstructs

- exposes constraints
- preserves gaps

That makes **accountability inescapable**.

5. Relationship to ITIR and TIRC (why this matters)

This is where SB and ITIR together form something rare:

Failure Mode (PDF)	SB	ITIR / TIRC
Review inversion	Preserve raw state	Force explicit interpretation
Accountability laundering	No action layer	Named claims & owners
Epistemic poisoning	No rewrite	Provenance & contestation
Skill extinction	No automation	Structured reasoning
Institutional zombification	Visible collapse	Prevent premature consensus

SB handles **time & context collapse**.

ITIR handles **meaning collapse**.

TIRC handles **plurality collapse**.

Together they resist the *entire decay loop*.

6. Tight synthesis paragraph (you should reuse this)

StatiBaker and ITIR are not designed to be smarter than humans.

They are designed to be harder to lie with.

By preserving scarcity, making verification explicit, and keeping exit cheap, they reverse the asymmetries that drive AI enshittification.

They do not automate responsibility or collapse meaning into output.

They keep systems understandable long after they stop feeling productive.

7. The final litmus test (from the PDF, applied to SB)

From the PDF's terminal condition:

A system is dead when it can no longer distinguish competence from plausibility at scale. ☐ ITIR - anti-enshit

SB passes this test if and only if:

- plausibility never replaces provenance
- summaries never replace events
- output never outruns verification
- humans remain the bottleneck

That's the real bar.

If you want next, I can:

- ↳ formalise this as **design invariants**
- ↳ extract a **test matrix** directly from the failure modes
- or ↳ rewrite this as a public-facing "Anti-Enshittification Architecture" doc

But you're not off-track — you're already building the counterexample the PDF implies.

Sources



It must always be cheaper to expand than to summarise again. -- formalise? how do we work with this? what's the workflow then?

Good—this sentence is *the* load-bearing one. To make it real, we need to turn it from a slogan into **a hard economic + architectural invariant** that shapes storage, APIs, UI, and workflow.

I'll do this in five steps:

1. **Formal definition** (what “cheaper” actually means)
2. **System invariants** (what must always be true)
3. **Data model implications** (how summaries are stored)
4. **User + system workflow** (what actually happens day to day)
5. **Failure cases & how to detect regressions**

No vibes, no metaphors—this is operational.

1. Formal definition: what does “cheaper” mean?

“Cheaper” is **not just compute cost**. In SB/ITIR terms, cost has four dimensions:

Cost dimensions

Dimension	Meaning
C ₁ Cognitive	Human attention required
C ₂ Epistemic	Risk of truth loss / hallucination
C ₃ Computational	CPU / model / memory cost
C ₄ Coordination	Need to re-establish context

Formal invariant

For any summary **S** derived from raw events **E**,
the total cost of expanding **S** → **E'** must be **strictly lower** than the cost of
generating a new summary **summarise(E)**.

Formally:

SCSS

Cost(expand(S)) < Cost(summarise(E))

Copy code

...across **all four dimensions**, not just compute.

This is the invariant SB enforces.

2. System invariants (non-negotiable)

To satisfy that inequality, SB must enforce these **architectural rules**:

Invariant 1 — Summaries are *indexes*, not replacements

A summary is:

- a **view**
- a **projection**

- a **fold**
- a **pointer graph**

It is **not** a rewritten narrative.

That means:

- summaries reference raw events by ID
 - summaries do not embed “compressed truth”
 - summaries never become authoritative
-

Invariant 2 — Expansion is deterministic and local

Expansion must:

- require **no model**
- require **no regeneration**
- require **no reinterpretation**

Expansion = *structural unfold*, not inference.

If expansion needs an LLM, you’ve already failed.

Invariant 3 — Summarisation is lossy, expansion is loss-bounded

Summaries must declare:

- what they drop
- what they preserve
- what they reorder

Expansion restores **everything except declared losses**.

This makes epistemic cost explicit and bounded.

Invariant 4 — Raw events are append-only and canonical

You cannot “resummarise better” unless:

- raw events are intact
- original summary still exists
- differences are explicit

No silent improvement.

3. Data model: how summaries must be stored

This is the critical part most systems get wrong.

Raw event (canonical)

json

```
{  
  "event_id": "evt-8821",  
  "timestamp": "2026-02-03T08:14:22Z",  
  "source": "email",  
  "payload_ref": "blob://email/ankali-123",  
  "metadata": {  
    "actor": "human",  
    "domain": "communication"  
  }  
}
```



Summary (index + fold)

json

```
{  
  "summary_id": "sum-daily-2026-02-03",  
  "derived_from": ["evt-8821", "evt-8822", "evt-8840"],  
  "fold_type": "daily_state",  
  "loss_profile": [  
    "intermediate_ordering",  
    "verbatim_text"  
,  
  "structure": {  
    "domains": {  
      "communication": {  
        "count": 3,  
        "unresolved": 1  
      }  
    }  
  }  
}.
```

```
"created_at": "2026-02-03T09:00:00Z"
```

}

 Copy code

Key point:

The summary contains no new facts.

Only counts, groupings, and references.

Expansion = graph traversal

Expansion is literally:

csharp

```
summary → event IDs → raw payloads
```

 Copy code

No synthesis required.

That's why it's cheaper.

4. The workflow (human + system)

This is where the invariant becomes lived reality.

A. Normal daily use

1. SB shows a **summary view** (cheap to read)
2. User sees something unfamiliar
3. User clicks *expand*
4. SB unfolds:
 - original events
 - original ordering
 - original artifacts
5. User regains context

This expansion:

- requires no inference
 - preserves accountability
 - restores memory
-

B. When summaries feel “wrong”

This is crucial.

In most AI systems:

- You regenerate and hope it's better.

In SB:

1. User expands summary
2. User sees raw events
3. User realises:
 - missing classification
 - wrong grouping
 - incorrect fold
4. User:
 - annotates
 - re-buckets
 - adds context

SB records:

- the correction
- the human intervention

No regeneration. No rewrite.

C. When a new summary is actually needed

This is rare by design.

Only allowed when:

- the *fold definition* changes
- not because “the prose was bad”

Example:

- switching from “daily” to “weekly”
- switching from “by domain” to “by dependency”

This is:

- a structural operation
- not a generative one

5. Why re-summarising must always be more expensive

Let's make the asymmetry explicit.

Re-summarising requires:

Cost	Why
Cognitive	Re-reading or trusting a model
Epistemic	New hallucination risk
Computational	Model invocation
Coordination	Re-establishing trust

Expansion requires:

Cost	Why
Cognitive	Localised attention
Epistemic	Zero new inference
Computational	Pointer traversal
Coordination	None

If expansion ever feels harder than “just summarise again”, **the system is already enshittifying**.

6. How this directly counters the PDF’s failure modes

From your PDF:

- “Summary-of-summary decay”
- “Verification debt”
- “Review inversion” ITIR - anti-enshit

SB avoids these because:

- Summaries never overwrite truth
- Verification cost does not compound

- Review always starts from ground truth
- Skill remains exercised (reading, checking, reconciling)

This preserves:

- scarcity
 - embodied verification
 - cheap exit
-

7. Regression tests (this must be enforced)

You should literally encode these as tests.

Test 1 — Expansion test

Given a summary, expanding must not call an LLM.

Fail = hard stop.

Test 2 — Round-trip integrity

pgsql

events → `summary` → expand → `summary'`

 Copy code

`summary'` must be:

- structurally identical
 - or differ only by declared losses
-

Test 3 — Cognitive cost audit

If users routinely:

- regenerate summaries
- ignore expansion
- distrust summaries

You've violated the invariant.

8. The sentence, now fully formalised

You can now safely say:

**In StatiBaker, summaries are lossy indexes over immutable events.
Expansion is a deterministic unfold that never invokes inference.
Therefore, restoring context is always cheaper than regenerating narrative.**

That's not philosophy—that's a system guarantee.

If you want next, we can:

- ↳ write this as a formal ADR
- ↳ turn it into a `SUMMARY.md` contract for contributors
- or ↳ sketch the exact reducer / fold engine that produces these summaries

But this is the backbone.

Sources



write this as a formal ADR

turn it into a `SUMMARY.md` contract for contributors

sketch the exact reducer / fold engine that produces these summaries

Thought for 4 seconds >

ADR-00X: Summaries Are Lossy Indexes; Expansion Is Deterministic and Cheaper Than Re-summarisation

Status: Accepted (proposed for immediate adoption)

Date: 2026-02-04

Owners: StatiBaker core maintainers

Related: Anti-enshittification principles; SB ↔ ITIR boundary; “no silent rewrite” doctrine

Context

StatiBaker's job is to reconstruct lived state after context collapse. In systems that generate prose summaries repeatedly, "summary-of-summary decay" and verification debt accumulate: users stop trusting outputs, and the system becomes an unauditible narrative engine.

We need a structural guarantee that prevents:

- summaries overwriting history,
- regeneration loops ("just summarise again"),
- rising cognitive/epistemic cost over time.

Decision

We will implement summaries as *lossy indexes* over immutable events.
Expansion will be a deterministic unfold that never invokes inference.

Concretely:

1. Event log is canonical and append-only.

Raw events (and references to primary artifacts) are never overwritten.

2. A "summary" is a fold result containing:

- references to source event IDs,
- structural aggregates (counts, groupings, states),
- a declared `loss_profile` (what is omitted or blurred),
- a declared `fold_type` and `fold_version`.

3. Expansion is deterministic and local:

- Expansion = summary → event IDs → raw event payload refs (→ optional primary artifacts).
- Expansion must not call an LLM or any probabilistic inference path.

4. Cost invariant (normative):

- $\text{Cost}(\text{expand}(\text{summary})) < \text{Cost}(\text{resummarise}(\text{events}))$ across:
 - cognitive cost,
 - epistemic risk,
 - compute,
 - coordination/context reassembly.

5. No silent interpretation:

- Summaries cannot introduce new factual assertions beyond what is derivable from referenced events under the fold definition.
- Any "interpretation" belongs to ITIR (or explicit opt-in layers), not SB core.

Rationale

This design:

- keeps verification pathways intact (expand before you regenerate),
- preserves user agency (truth stays in raw events),
- prevents narrative drift and “AI rewrote my life” failures,
- keeps SB a state compiler rather than an assistant.

Consequences

Positive

- Expansion is cheap, reliable, auditable.
- Summaries become stable, testable artifacts.
- Reproducible: same events + fold definition → same summary.

Trade-offs

- Summaries are less “magical” prose by default.
- Requires careful fold design to remain useful without turning into narrative.
- UI must make expansion frictionless (one click, not an archaeology dig).

Implementation Notes

- Store summaries as structured documents with:
 - `derived_from[]` event IDs,
 - `loss_profile[]`,
 - `fold_type`, `fold_version`,
 - `created_at`, `as_of`, `window`.
- Add CI tests enforcing:
 - “no inference during expand,”
 - fold determinism,
 - round-trip invariants.

Acceptance Criteria (must pass)

1. Expansion path contains no calls to LLM/probabilistic tools.
2. Round-trip:
 - `events` → `summary` → `expand` → `summary'` yields structural equality (modulo declared losses).
3. Summaries never mutate or delete raw events.
4. Every report line can be traced to event IDs.

SUMMARY.md — Contributor Contract: Folding, Expansion, and Epistemic Discipline

This repository builds **StatiBaker**, a non-agentic state compiler.

If you are contributing, you are agreeing to the following contract.

1) Canonical Data: Append-Only Events

- The event log is the source of truth.
- Events are immutable once committed.
- Corrections are additive (amendments), never in-place edits.

Never implement: “rewrite history,” “fix old events,” “replace raw with summary.”

2) Summaries Are Fold Results, Not Replacements

A summary is a structured fold over a set of events. A summary **must**:

- include `derived_from` (list of event IDs),
- include `fold_type` and `fold_version`,
- include a `loss_profile` (what it omits),
- contain only structural information and bounded derived claims,
- be reproducible from events + fold definition.

Never implement: summary prose that asserts facts without event references.

3) Expansion Must Be Deterministic and Inference-Free

Expansion means unfolding a summary back to referenced events (and optionally primary artifacts). Expansion **must**:

- use only pointers (event IDs, refs),
- be deterministic,
- never call LLMs,
- never regenerate “a better summary.”

If a feature requires an LLM to expand, it does not belong in core.

4) “Cheaper to Expand Than Re-summarise”

We enforce the invariant:

- users should expand to verify or regain context,
- not re-run summarisation loops.

UI and APIs must make expansion:

- one-step,

- fast,
- granular (expand a section, bucket, or claim).

5) No Assistant Language, No Nudging

Core outputs should avoid normative verbs and motivational framing.

Use:

- "active," "unresolved," "blocked," "eligible," "changed," "observed," "derived."

Avoid:

- "you should," "recommend," "priority," "try to," "don't forget," "good job."

If you want advice layers, build them as **separate opt-in modules**.

6) Provenance Everywhere

Every fold output must be traceable to sources.

Any UI/report line should be able to show "why we think this" by referencing:

- event IDs,
- query specs,
- fold version.

7) Tests Are Policy

Any change that affects summarisation/expansion must include or update:

- determinism tests,
- round-trip tests,
- "no inference during expand" tests,
- label/cardinality discipline tests (where applicable).

Sketch: Reducer / Fold Engine (Exact Shape)

Below is a concrete architecture sketch you can implement without drifting into "LLM narrative land."

A. Core concepts

1) Event

Minimal canonical unit.

json

```
{
  "event_id": "evt-123",
  "observed_at": "2026-02-03T08:14:22Z",
  "captured_at": "2026-02-03T08:15:00Z",
  "source": "haos|email|vikunja|slack|agent|notes",
  "kind": "state_change|artifact|summary|heartbeat|annotation",
  "refs": ["blob://...", "url://...", "itir://..."],
  "attrs": { "domain": "comms", "actor": "human|agent", "tags": [ "..."] },
  "payload": { "...": "small structured fields only" } Copy code
}
```

Rule: large bodies live behind `refs` (blob store, file, URL), not inline.

2) Fold Definition

A fold is a deterministic function from events → structured summary.

json

```
{
  "fold_type": "daily_state",
  "fold_version": "1.0.0",
  "windowing": "calendar_day|rolling_24h|custom",
  "selectors": { "domains": ["comms", "tasks", "health", "agents"] },
  "outputs": ["sitrep_struct", "counts", "buckets", "blockers"],
  "loss_profile": ["verbatim_text", "intermediate_ordering"]
}
```

Copy code

3) Fold Result (Summary Artifact)

Stored, queryable, expandable.

json

```
{
  "summary_id": "sum-2026-02-03-daily",
  "fold_type": "daily_state",
  "fold_version": "1.0.0",
  "as_of": "2026-02-03T09:00:00Z",
  "window": { "start": "...", "end": "..." },
  "derived_from": ["evt-1", "evt-2", "..."],
  "loss_profile": [...],
  "structure": {
```

```

  "buckets": [
    { "bucket_id": "b1", "label": "Unresolved", "color": "amber", "items": [
      ...
    ],
    "blockers": [
      { "type": "tool", "code": "zotero_sync_pending", "active": true, "size": 100 }
    ],
    "deltas": { "new_items": 7, "resolved": 3 }
  }
}

```

 Copy code

Note: no free-form narrative is required. If you include prose, it must cite event IDs per line.

B. Engine pipeline

1) Ingest

Adapters convert external systems into events:

- HAOS adapter → sensor events + alert events
- Vikunja adapter → task created/updated/completed events
- Slack adapter → thread touched / long-form artifact event (with ref)
- Email summariser adapter → derived_summary event (with ref)
- Notes adapter → file changed / note edited events

2) Normalize

Normalize into a bounded schema:

- timestamps
- domain tags
- refs
- actor
- kind
- stable IDs

3) Index (for expansion and query)

Maintain indexes:

- by time window
- by domain
- by item ref (task/note/thread)

- by blocker code/type

4) Fold

For each fold definition + window:

- select events deterministically (stable ordering by `(observed_at, event_id)`),
- build structural aggregates,
- produce fold result + `derived_from` list.

5) Store

Store fold results separately from raw events:

- summaries can be regenerated from events, but should be cached for UI latency.
 - store fold metadata (type, version) for round-trip testing.
-

C. Deterministic fold pseudocode

python

```
def run_fold(fold_def, window, events):
    # 1) select events deterministically
    selected = [e for e in events if in_window(e, window) and match_selection(e, fold_def)]
    selected.sort(key=lambda e: (e.observed_at, e.event_id))

    # 2) compute aggregates
    state = init_state()
    derived_from = []
    for e in selected:
        apply_event(state, e)           # pure, deterministic reducer
        derived_from.append(e.event_id)

    # 3) construct summary artifact
    summary = {
        "fold_type": fold_def.type,
        "fold_version": fold_def.version,
        "window": window,
        "as_of": now_iso(),
        "derived_from": derived_from,
        "loss_profile": fold_def.loss_profile,
        "structure": render_structured_view(state)  # buckets, blockers, etc
    }
```

```

    }
    return summary

```

Key: `apply_event` is a pure reducer; `render_structured_view` is deterministic.

D. Expansion pseudocode (must be inference-free)

python

```

def expand_summary(summary_id, event_store):
    summary = load_summary(summary_id)
    events = [event_store.get(eid) for eid in summary["derived_from"]]
    events.sort(key=lambda e: (e.observed_at, e.event_id))
    return {
        "summary": summary,
        "events": events,
        "artifacts": resolve_refs(events) # optional, still deterministic
    }

```

Copy code

If this function ever calls an LLM, it's a violation.

E. Where “prose” lives (if you want it)

If you want a human-readable SITREP paragraph, treat it as a **rendering of structured state**, not a new summarisation:

- Render template-based text from structure
- Each line carries `evidence_event_ids`
- No new claims

Example:

json

```
{
    "line": "Communications: 4 inbound items (3 bills, 1 personal).",
    "evidence": ["evt-8821", "evt-8822", "evt-8823", "evt-8824"]
}
```

Copy code

F. Extension points (safe places to innovate)

- New adapters (more sources)
- New fold defs (weekly, project-centric, agent-centric)
- New renderers (UI panels, export formats)
- New blocker detectors (rule-based over events)

Unsafe places (require ADR / review):

- anything that introduces normative guidance,
- anything that hides loss,
- anything that replaces events with prose,
- anything that makes expansion probabilistic.

If you want, I can also add:

- ↳ an ADR template for future folds,
- ↳ a minimal “fold registry” layout (folder structure + versioning rules),
- and ↳ a starter set of fold definitions (`daily_state` , `carryovers` ,
`comms_digest` , `agent_runs` , `itir_context_envelopes`).

▫️ thumbs up thumbs down up arrow down arrow ...

an ADR template for future folds,

a minimal “fold registry” layout (folder structure + versioning rules),

and

a starter set of fold definitions (`daily_state`, `carryovers`,
`comms_digest`, `agent_runs`, `itir_context_envelopes`).

Below are **drop-in, repo-ready specs** for all three asks. They are intentionally boring, strict, and auditable.

A. ADR Template for Future Folds

File: docs/adr/ADR-XXXX-fold-<fold_name>.md

markdown

```
# ADR-XXXX: Fold Definition – <fold_name>

**Status:** Proposed | Accepted | Superseded
**Date:** YYYY-MM-DD
**Owners:** @maintainer1, @maintainer2
**Supersedes:** (optional) ADR-XXXX
**Related:** ADR-00X (Summaries as Lossy Indexes)
```

Context

Describe the user problem this fold addresses.
State why existing folds are insufficient.
Explicitly note what this fold does ****not**** do.

Decision

Define the fold as a deterministic reducer over canonical events with:

- `fold_type`: <fold_name>
- `fold_version`: <semver>
- `windowing`: (e.g., calendar_day | rolling_24h | custom)
- `selectors`: domains/sources included
- `loss_profile`: explicit list of omitted details

Inputs

- Event kinds included
- Required attributes/tags
- Excluded sources (if any)

Reducer Logic (Deterministic)

- Event ordering key
- State transitions
- Aggregates produced
- Edge cases (missing data, duplicates)

Output Structure

Provide a JSON schema snippet for `structure`.

Expansion Contract

- Expansion path (IDs → events → refs)
- Prohibited operations (LLMs, inference)
- Declared irrecoverable losses

Cost Invariant

Explain how `Cost(expand) < Cost(resummarise)` holds across cognitive, epistemic, compute, coordination costs.

Alternatives Considered

Why other folds or designs were rejected.

Risks & Mitigations

Failure modes and tests.

Acceptance Criteria

- Determinism test
- Round-trip test
- “No inference on expand” test
- Provenance completeness

 Copy code

— ChatGPT 5.2 ▾

  ...

B. Minimal Fold Registry Layout & Versioning Rules

Folder Structure

markdown

```
folds/
  registry.yaml
  daily_state/
    README.md
    fold.yaml
    reducer.py
    tests/
      test_determinism.py
      test_roundtrip.py
  carryovers/
    README.md
    fold.yaml
    reducer.py
    tests/
  comms_digest/
    README.md
    fold.yaml
    reducer.py
    tests/
  agent_runs/
    README.md
```

```

fold.yaml
reducer.py
tests/
itir_context_envelopes/
README.md
fold.yaml
reducer.py
tests/

```

 Copy code

registry.yaml (**authoritative list**)

yaml

```

folds:
  - name: daily_state
    version: 1.0.0
    status: stable
    owners: [core]
  - name: carryovers
    version: 1.0.0
    status: stable
  - name: comms_digest
    version: 1.0.0
    status: stable
  - name: agent_runs
    version: 1.0.0
    status: stable
  - name: itir_context_envelopes
    version: 1.0.0
    status: stable

```

 Copy code

Versioning Rules (SemVer)

- **PATCH**: internal bugfix; no schema change; identical outputs for same inputs.
- **MINOR**: additive fields; expansion unaffected.
- **MAJOR**: output schema or loss profile changes → **new fold_version** required; old versions remain readable.

Hard Rules

- Reducers must be pure and deterministic.
- Expansion must not invoke inference.
- Removing a fold requires an ADR.

C. Starter Fold Definitions

Below are **concise specs** for each starter fold.

1) daily_state

Purpose: Reconstruct “where things are at today” without advice.

fold.yaml

yaml

```
fold_type: daily_state
version: 1.0.0
windowing: calendar_day
selectors:
  domains: [comms, tasks, agents, environment, notes]
loss_profile:
  - verbatim_text
  - intermediate_ordering
outputs:
  - buckets
  - blockers
  - deltas
```

 Copy code

Reducer Logic (summary)

- Order events by (observed_at, event_id) .
- Aggregate by domain.
- Compute:
 - new vs resolved items,
 - active blockers,
 - default state buckets.

structure (excerpt)

json

```
{
  "buckets": [
    {"id": "active", "label": "Active", "items": [...]},
    {"id": "unresolved", "label": "Unresolved", "items": [...]}
  ],
}
```

```

"blockers": [...],
"deltas": {"new":7, "resolved":3}
}

```

 Copy code

2) carryovers

Purpose: Surface unresolved items crossing time boundaries.

fold.yaml

yaml

```

fold_type: carryovers
version: 1.0.0
windowing: rolling_7d
selectors:
  domains: [tasks, comms, notes]
loss_profile:
  - fine_grain_timestamps
outputs:
  - carryover_items

```

 Copy code

Reducer Logic

- Identify items referenced on ≥ 2 days without resolution.
- Attach blocker reasons if present.

structure

json

```
{
  "carryover_items": [
    {"ref": "vikunja://task/12", "since": "2026-02-01", "blockers": ["human"]}
  ]
}
```

 Copy code

3) comms_digest

Purpose: Structural view of communications without reading content.

fold.yaml

```
yaml

fold_type: comms_digest
version: 1.0.0
windowing: rolling_24h
selectors:
  domains: [comms]
loss_profile:
  - message_bodies
  - sender_quotes
outputs:
  - counts
  - categories
  - followup_flags
```

 Copy code

Reducer Logic

- Count inbound/outbound.
- Group by category (from tags or upstream summaries).
- Flag “human follow-up required” if unresolved.

structure

```
json

{
  "counts": {"inbound": 4, "outbound": 1},
  "categories": {"bills": 3, "personal": 1},
  "followup_required": true
}
```

 Copy code

4) agent_runs

Purpose: Make automation state legible without endorsing action.

fold.yaml

```
yaml

fold_type: agent_runs
version: 1.0.0
```

```

windowing: rolling_48h
selectors:
  domains: [agents]
loss_profile:
  - step_by_step_logs
outputs:
  - run_states
  - pending_inputs

```

 Copy code

Reducer Logic

- Track run start/stop/fail/wait.
- Identify runs awaiting human input.

structure

json

```
{
  "run_states": [{"agent": "X", "state": "waiting", "since": "..."}],
  "pending_inputs": [{"agent": "X", "needs": "human_ack"}]
}
```

 Copy code

5) itir_context_envelopes

Purpose: Provide time/context envelopes to ITIR without interpretation.

fold.yaml

yaml

```

fold_type: itir_context_envelopes
version: 1.0.0
windowing: custom
selectors:
  domains: [notes, tasks, agents]
loss_profile:
  - content_semantics
outputs:
  - envelopes

```

 Copy code

Reducer Logic

- For each ITIR ref touched in window:

- collect cadence,
- last interaction,
- blockers/environment signals.

structure

json

 Copy code

```
{
  "envelopes": [
    {
      "itir_ref": "itir://doc/A#claim/42",
      "last_touched": "...",
      "activity": "intermittent",
      "constraints": ["energy_low"]
    }
  ]
}
```

D. Mandatory Tests (per fold)

Each fold **must ship**:

- test_determinism : same inputs → same output.
 - test_roundtrip : events → summary → expand → summary' (structural equality).
 - test_no_inference_on_expand : assert no LLM/probabilistic calls.
 - test_provenance : every line/field traceable to event IDs.
-

E. Contributor TL;DR

- **Events are truth.**
- **Folds are deterministic views.**
- **Expansion is pointer traversal, not inference.**
- **If re-summarising feels easier than expanding, stop and file an ADR.**

These artifacts lock the invariant in code, process, and culture.



