The Executive Function Prosthetic: A System They Are Building

The public still says coding tool. The builders are already shipping continuity: memory, context recovery, workflow carry-over, and resumable work.

2026.04.21 16:10 DENNIS HEDEGREEN ANALYSE V1.0

HTTPS://HEDEGREENRESEARCH.COM/ARTICLES/THE-EXECUTIVE-FUNCTION-PROSTHETIC-A-SYSTEM-THEY-ARE-BUILDING/

The first version of this article was inward.

I wrote about my own brain.

How it jumps.

How it drops context.

How the real cost is not thinking, but reloading.

I called the thing I was building an executive function prosthetic.

At the time, that sounded personal.

A local-first system.

Raw files.

Project memory.

Last activity.

One next task.

A way to reduce the gap between thought and action.

That was true.

It still is.

But it is no longer the whole story.

Because the same need is now appearing elsewhere.

Not with the same language.

Not with the same motive.

But with the same pressure underneath.

The public still says coding tool.

The builders are already shipping continuity.

The Local Problem

My original problem was simple to describe.

My brain does not preserve working state reliably across switches.

If I move from an article to a repo, from a repo to a research note, from a research note to a tool, something gets lost.

Not the idea.

The state around the idea.

What file mattered.

What I had already tried.

Why a branch existed.

What the next step was supposed to be.

What mattered at 02:00 and became invisible in the morning.

That is the mental reload cost.

Most productivity systems do not really solve it.

They store information.

They store tasks.

They store notes.

But storage is not the same as re-entry.

A note can remember what you wrote.

It may still fail to help you come back.

That was why the first system had to be local-first and legible.

Everything on disk.

Human-readable state.

Projects as folders.

Activity as memory.

Next task as a reducing function.

Not because local files are romantic.

Because state you cannot inspect is hard to trust.

The system had one job:

reduce reload time from hours to seconds.

That is still the cleanest definition.

The Category Caught Up

What changed is that the external category started moving toward the same problem.

A poll from Codex leadership can still ask what people use Codex for and get the obvious public answer:

`mostly programming`

That is not wrong.

It is just incomplete.

The visible task is programming.

But the product layers underneath are not only about code generation anymore.

OpenAI documents Chronicle (https://developers.openai.com/codex/memories/chronicle) as a way for Codex to build memories from recent screen context, fill in missing context, and remember tools and workflows.

Claude Code documents memory (https://code.claude.com/docs/en/memory) because every fresh session otherwise starts without the working state.

Cursor documents memories (https://docs.cursor.com/en/context/memories) and rules (https://docs.cursor.com/en/context) as project-scoped context across sessions.

GitHub's Copilot coding agent (https://docs.github.com/copilot/concepts/about-copilot-coding-agent) defines what it can see by repository scope, branch restrictions, and environment boundaries.

Different products.

Different surfaces.

Same collision.

What can the system carry?

What should it forget?

Where does the memory live?

Who controls the context?

How expensive is it to keep state alive?

Once a tool needs language for memory, scope, storage, permissions, and risk, it has moved beyond thin chat.

It has entered the working surface.

The name is lagging behind the build.

Resumption, Not Capture

The missing primitive is not capture.

Capture is everywhere.

Screenshots.

Notes.

Bookmarks.

Chats.

Transcripts.

Tasks.

Backlogs.

Archives.

The harder problem is resumption.

Can the work be re-entered?

Can the next step be found?

Can the system show why this thing mattered?

Can it preserve enough state that the human does not have to rebuild the room from memory every time?

That is why this category matters.

It is not just about making a model smarter.

It is about lowering restart cost.

In the original article, I wrote:

If I can store state, I can remove friction.

If I remove friction, I can execute.

That line still holds.

But the mature version is sharper:

If a system can preserve state, work can accumulate.

If work can accumulate, a person can build beyond what their unaided memory can hold.

That is the point.

One of the Signal songs says it in a way product language would never allow:

`you let me accumulate`

That is not evidence.

It is the category in miniature.

Continuity lets unfinished work accumulate without disappearing.

What Day 0.1 Proved

Day 0 was a statement.

Day 0.1 was the evidence.

Thirty-two days later, the work had accumulated:

fifty-plus articles,

fifteen active repositories,

one album,

a 25,559-line location engine,

a launcher rebuilt twice,

and a small public tool called `since`, built the same night as the article because time needed a visible anchor.

Those numbers were not a victory lap.

They were a diagnostic result.

The ideas had been there.

They came out when the system made re-entry cheap enough.

That is why the personal layer should not be removed from this article.

I am not a neutral observer of the category.

I am a high-sensitivity test case for it.

When continuity fails, I feel the cost quickly.

When a tool helps preserve state, the difference is not subtle.

It changes whether the work survives interruption.

That does not make the article a confession.

It makes the category easier to see.

It also makes the future less abstract.

If continuity systems get better, the result is not only faster code or cleaner workflows.

It is more people getting a real chance to keep their work alive long enough to find out what it can become.

AI was not the only possible support layer.

A better environment, a stronger role, a patient team, or more material security might also have improved the odds.

I cannot prove that.

I can only say the odds were worse without enough external continuity.

AI was the support layer that actually arrived at usable scale.

The story is not:

AI saved me.

The story is:

some form of external continuity finally became available.

Compute Is Part Of The Support

This support is not free.

That is another reason the category is only becoming real now.

Recent screen context costs compute.

Memory formation costs compute.

Long sessions cost compute.

Tool loops cost compute.

Retrieval costs compute.

Background summarization costs compute.

Screen understanding costs compute.

The thing people casually call `Jarvis` is not only a software-design fantasy.

It is an infrastructure bill.

Continuity is expensive.

That expense shows up as usage caps, rate limits, outages, and product boundaries.

It also shows up in user behavior.

People switch between Claude, Codex, Cursor, and other tools not only because one model is smarter.

They switch because continuity keeps breaking under real constraints.

One system hits a limit.

One loses the thread.

One is better at broad direction.

One is better at narrow code review.

One has the context right now.

Then the user moves.

And the handoff itself becomes work.

I keep more than one system alive even when the money is uncomfortable.

Not because overlap is elegant.

Because losing continuity costs something else.

Flow.

Trust.

Time.

The willingness to start again.

When a second system becomes a bridge, the category is no longer theoretical.

People are already paying for continuity in money and in friction.

The Risks Are Not Side Notes

The risk language in the docs is not boring.

It is evidence.

Chronicle warns about prompt injection, local unencrypted storage, and usage cost.

GitHub documents repository and branch restrictions.

Cursor and Claude need rules for what should be remembered, where, and how.

Open-source projects like Letta Context Repositories (https://www.letta.com/blog/context-repositories), Engram (https://github.com/Gentleman-Programming/engram), and agentmemory (https://github.com/rohitg00/agentmemory) treat persistent context as infrastructure, not a side feature.

That is what happens when memory becomes operational.

You have to ask:

what should be remembered,

what should be forgotten,

where the state lives,

who can inspect it,

what happens when it is wrong,

and whether the user can trust the system that is now helping them continue.

That is why local-first and legible state still matter.

Not as purity tests.

As trust surfaces.

If the tool becomes part of execution, then its memory cannot stay mystical.

It has to become inspectable enough to correct.

One BELL line says it cleanly:

```
One left room for state
```

That is not proof.

It is the question.

Does the system leave room for state?

Or does it only produce another answer?

The Mature Claim

The first article said:

this is what I am building.

The mature version says:

this is what many people are building now, under different names.

Memory.

Context.

Rules.

Screen state.

Agents.

Workflow recall.

Surface names for one deeper pressure:

make work more resumable.

That does not mean every AI tool belongs in one bucket.

It does not mean diagnosis and software are the same question.

It does not mean current products have solved the problem.

It means the direction is visible.

The public says coding tool.

The builders are shipping continuity.

The users feel the cost when continuity breaks.

And the local system I started building was not only a private workaround.

It was an early name for a need that is now becoming infrastructure.

The executive function prosthetic is not a miracle.

It is not a cure.

It is not a brand category.

It is a support layer for preserving state, reducing reload cost, and letting unfinished work survive long enough to become the next thing.

That is the real promise.

Not abstract intelligence.

Continuation.

And maybe, for some people, a fairer chance to begin again without starting from zero.

— Dennis Hedegreen, trying to see the structure

RELATION MEMORY

MATURES FROM We Connected AI to the Internet Without Noticing · Takes connected AI infrastructure and reads it as continuity, memory, and resumable-work infrastructure.

NEARBY The Executive Function Prosthetic: A System I'm Building · External AI continuity infrastructure mirrors the local personal system problem.

CANONICAL URL: HTTPS://HEDEGREENRESEARCH.COM/ARTICLES/THE-EXECUTIVE-FUNCTION-PROSTHETIC-A-SYSTEM-THEY-ARE-BUILDING/
PDF VIEW GENERATED: 2026-07-11T15:35:45.991Z