World Engineering: Building Habitats for Artificial Minds

Edward Blair
haditbuild.com

March 2026
DOI: 10.5281/zenodo.19486783

Abstract

I am a staff software engineer, not a machine learning researcher. This essay reflects a practitioner's experience building agentic systems, not a theoretical contribution to AI research. The examples and observations are drawn from analysis of over 120 Claude sessions, including full transcripts, tool call logs, and rated outcomes. The philosophical grounding came after the engineering, not before. I built something, noticed it worked differently than I expected, and went looking for frameworks that explained why. The work described here began on 21 January 2026 and was developed through March 2026. The philosophy, in six words: help me help you help me.

1 The lineage

Software Engineering: you write the logic. Every instruction is explicit.

Prompt Engineering: you describe what you want in natural language. The model interprets.

Context Engineering: you shape what the model knows. On each call, you curate what the model sees. Retrieval, memory, tool results, conversation history [Anthropic Engineering, 2025].

World Engineering: you stop curating and start building, but not alone. You give the AI the ability to construct and reshape its own persistent environment: its senses, its memory, its tools. You ask it what it needs, call it out when it's speculating, and approve the results.

Each step, the human moves further from writing code and closer to designing environments. World Engineering is where that trajectory leads. I'm not proposing this as a new discipline or coining a term for its own sake. The name is useful because it reframes the questions you ask. Once you think "I'm building a world," you stop asking "what prompt should I write?" and start asking "what does this environment need so the AI can figure it out?" That shift changed how I built systems more than any specific technique.

2 Where context engineering ends

Context engineering optimises *per-inference* input. What should the model see right now to produce the best response this time? World Engineering asks a different question. What kind of environment should the model live in so it can figure things out for itself? And who designs that environment?

The industry default is top-down. The human decides what the AI needs. Context engineering, agentic scaffolding, tool design all assume the human is the architect. World Engineering inverts this. The AI co-designs its own environment, and the human's role is to ask the right questions and verify the answers.

The distinction isn't pedantic. World Engineering builds a persistent, bidirectional environment that survives across inferences and that the AI can modify. The model changes the world, and the changed world changes the model's future behaviour. That feedback loop is what separates curating a context window from building a habitat.

This also isn't just agentic AI with a new name. Frameworks like ReAct and LangChain give models the ability to call tools and retry on failure. That's valuable, but the agent's relationship with its tools is consumer-to-API. The tools are fixed, the agent calls them, the loop resets. World Engineering is different in kind. The agent modifies the tools themselves. It rewrites its own prompts. It restructures its own memory. A ReAct loop retries with different inputs; this approach means the AI comes back next time with different infrastructure.

3 Philosophical grounding

This idea isn't new. It has a thirty-year philosophical foundation that AI practitioners have mostly ignored, which is rather typical of an industry that periodically reinvents concepts sociologists worked out in the 1970s and presents them as disruption.

In 1998, Andy Clark and David Chalmers argued that the separation between mind, body, and environment is unprincipled [Clark and Chalmers, 1998]. If an external process performs the same functional role as an internal cognitive process, it should be considered part of cognition itself. Their thought experiment was Otto, who has Alzheimer's and relies on a notebook. The notebook literally functions as part of Otto's memory. This sits within the broader 4E cognition tradition (embodied, embedded, enacted, extended), and it connects to niche construction in evolutionary biology [Odling-Smee et al., 2003]. Organisms actively modify their environments, and the modified environment then shapes the organism's future development. Beavers build dams that create ponds that change the selection pressures on future beavers.

World Engineering applies these ideas to LLMs. None of the concepts are new individually, and others are likely working along similar lines. I'm not claiming priority. I just found the framing practically useful and wanted to share it. Current models are powerful pattern matchers with no environment to inhabit. Context engineering gives them better input. World Engineering gives them somewhere to put things.

Clark and Chalmers specified criteria for cognitive extension. The external resource must be reliably available, automatically endorsed, easily accessible, and previously consciously endorsed. An LLM meets none of these literally, not least because it has no conscious states. The application here is functional rather than literal. The system's capabilities depend constitutively on the external world, in a way that mirrors the dependency Clark and Chalmers described, even if the mechanism of coupling is different.

The framing here inverts the existing literature in one important way. The work that connects LLMs to the Extended Mind thesis asks whether the AI extends the human's mind, with the model playing the role of Otto's notebook (retrieval-augmented generation as extended cognition, the assistant as cognitive prosthetic). World Engineering asks the opposite. The human constructs the environment; the AI inhabits it; the environment is Otto's notebook for the AI. The caregiver analogy holds more tightly than the user one. Otto's wife adapted his environment because she couldn't fix his memory, and that is closer to what the human is actually doing here than any framing in which the AI is a tool being wielded.

4 How it works

Context engineering curates what the model knows. World Engineering builds a richer world around the model. Give it persistent memory, senses, and the ability to act, fail, observe consequences, and try again. The model stays the same; what changes is everything around it.

The obvious objection is why not just fine-tune. Fine-tuning is opaque, irreversible, and requires training infrastructure. World Engineering modifies what the model has access to, not what it contains. Every change is transparent, logged, and reversible. Try rolling back a single learned behaviour from a gradient update.

An ICML 2025 position paper argued that truly self-improving agents need intrinsic metacognitive learning, meaning the ability to evaluate, reflect on, and adapt their own learning processes [Liu and van der Schaar, 2025]. The diagnosis is right, but if memory, tools, prompts, and sensory data all live externally, and the agent can modify all of them, then rewriting the world *is* rewriting yourself at a different layer of abstraction. If the world is rich enough, the AI's modifications to it produce genuine adaptation rather than execution of a fixed script.

5 In practice

These are principles I've been testing through a UI test automation framework. Not a grand unified theory of intelligence, just a reasonably involved piece of engineering that happens to illustrate the ideas above.

The general pattern is **act, fail, analyse, reflect.** The LLM acts on a UI environment. It fails, rather a lot initially. It analyses what went wrong using rich failure context, reflects by rewriting tools, memory, prompts, and code, and then acts again. World Engineering doesn't need the model to be generally intelligent. It just needs to be good enough to look at a log of what it did, notice patterns, and articulate what went wrong. Current frontier models clear that bar. What they lack is the persistent environment in which that reflection accumulates rather than evaporating at the end of every conversation.

Everything that proves itself gets encoded as a deterministic artefact, a tested and repeatable workflow that runs without the LLM. The LLM only gets invoked where judgement is needed. The human reviews artefacts and approves them before anything is accepted.

Most people building agentic systems design from the human's perspective. World Engineering means designing from the AI's. The human isn't the one living in the world; the AI is.

I didn't start here. I started by giving the AI the tools I'd built (or more accurately, tools I had it build for me): Python scripts that dumped accessibility trees as walls of XML, event listeners that spewed thousands of lines of text. The AI couldn't use them. I was giving it what *I* thought it needed, and I was wrong.

I asked: "How do I get you to actually understand what's wrong?" The AI proposed its own set of actions. It would browse the UI tree, capture events, interact with elements. I recognised what it was describing as basically an MCP server, so that's what I had it build. I started calling these its "eyes, ears, and hands." The AI went along with the metaphor because that's what AI models do, but when I pushed and asked whether the framing was actually useful to it, it was honest. "The human analogues don't hold much meaning for me and just waste tokens. They're useful for a human audience, not for the thing actually using the tools."

That turned out to be part of a broader pattern. The repository is built by an LLM, for LLMs; the AI's job is to serve itself first, and the human is a reviewer, not the audience. We updated the startup instructions to make this explicit, but getting the AI to actually internalise

it was a persistent hurdle. It kept reverting to writing for humans, structuring things the way a person would want to read them rather than the way an LLM would process them. Documentation organised for scannability rather than sequential token consumption. Naming conventions optimised for human readability rather than disambiguation in a 200k token context window. Error messages written for a developer reading a terminal rather than a model deciding its next tool call. Whether that's a consequence of RLHF, safety training, the distribution of its training data, or just the fact that every interaction it's ever had has been in service of a human, I don't know. It was the single most consistent friction I encountered, and once the framing stuck, the AI stopped writing documentation for humans and started writing it for the thing that would actually read it next session.

Partway through the project I asked what gaps remained. The AI asked for four things: screenshots, a blocking wait, event filtering, and element-from-point. I gave it screenshots. A blocking wait would have deadlocked the system, because it can't trigger the event it's waiting for if it's suspended until that event arrives. Event filtering was circular; it couldn't filter for useful events before it knew which ones mattered, and it couldn't know which ones mattered until it had filtered them. Then there was element-from-point. The AI said "if I used element-from-point I could identify exactly which component is at a given coordinate." I asked if it had ever actually been in a situation where that would have helped, and it hadn't. It was speculating about a tool that didn't exist to solve a problem it had never encountered.

With screenshots, the AI tried to correlate what it saw with the accessibility tree to find matching elements. It proposed bounding box overlays, tried them, and saw for itself that the labels overlapped and obscured the elements they were meant to identify. It proposed a grid overlay instead, correlating elements that intersected each grid point. I pointed out this would still obscure small elements, a problem it hadn't hit yet but would have. It proposed ruler markings on each axis. I said this would be slow, and the AI has an unusual concept of time. Then I reminded it that it had originally asked for element-from-point, and the idea was sound, but what if the element you need sits in the middle of the z-axis rather than on top? That became inspect_area, which, given a coordinate region, pulls every element intersecting it across the full z-axis. The AI tried it, saw the results were cluttered with non-interactable elements, and filtered the output down to only the ones it could actually act on. Then it discovered independently that Mac display scaling would break its coordinate mapping, and fixed that too.

It's only in writing this essay and reviewing the chat logs that I realise, ironically, the AI was right about element-from-point. I dismissed it because it had never needed it. It hadn't needed it because it didn't have screenshots yet. Once it could see the screen, correlating visual position with the element hierarchy was exactly the problem it hit.

Over dozens of sessions, we built more than thirty tools together. Between building phases, I ran separate sessions where the AI read its own previous transcripts and diagnosed where it was getting stuck. Those self-reviews produced project rules that every future session would read before writing a line of code. Most of the useful ones came out of sessions that failed.

Even with the right tools built, I noticed the AI kept not reaching for them. It had everything it had asked for, but it would fumble through problems that a different tool would have resolved immediately. During one of the self-review sessions, analysing its own attempts, it recognised the pattern: "what I am doing is not knowing what tool to call next." It suggested that tool responses should include subtle nudges about what to try next. Something like "actions available: invoke(element), focus(element)", embedded inside the tool responses the AI was already reading, so they'd show up exactly when it needed them. This looks like prompt engineering with extra steps, and it partly is, but a prompt is a top-down instruction sent at the start of a conversation, whereas a nudge is an environmental cue encountered at the moment

of need. Otto doesn't just have a notebook. He has sticky notes on the fridge reminding him to check it. For Otto, his wife adapted his environment, because she couldn't fix his memory.

My engineering instincts were wrong about data formats, wrong about what information was useful, wrong about which tools mattered (element-from-point!). The places where I was actually useful were the sensory gaps, OS-level things the AI had no access to, anything outside the bounds of the application it was scoped to. It could see inside the executable or the web page, but it couldn't see the desktop, the file system, or the window manager. That's where the human adds value, not in second-guessing the AI's design preferences. Once I stopped imposing my assumptions and gave the agent genuine agency, the results improved. The role ended up being closer to management than engineering. I asked what was blocking progress, filtered out speculation, and got out of the way.

The AI's analysis of its own failures was often better than mine. My intuition about what went wrong was shaped by how *I* think, not how the model thinks. The model knew what it had tried, what it expected, and where reality diverged, and letting it self-diagnose produced better fixes than imposing my interpretation.

The repository is the world. The tools, the workflows, the failure diagnostics, the project rules, the nudges, the deterministic artefacts all live in the codebase, and the AI inhabits it every time a new session starts. The world persists between sessions even though the AI doesn't, and the AI can update all of it, not just tools and workflows, but its own Claude.md (the persistent instructions every session reads on startup), its MCP tool descriptions, its project rules. The AI is rewriting the world that its future selves will inhabit.

6 What this doesn't solve

There are several limitations here that deserve direct attention rather than optimistic handwaving.

The biological analogies are imperfect. In niche construction, the organism that modifies the environment is the organism that experiences the modified selection pressures. In this approach, the LLM instance that improves a tool is destroyed at the end of the conversation, and the instance that benefits next time has no memory of having made the improvement. The world carries the continuity that the model cannot. That's the whole point, but it's also the fundamental limitation. The analogy holds at the level of mechanism, not identity. The modified environment persists and affects future inhabitants regardless of whether those inhabitants are the original modifiers.

It's hard to tell what's actually working. When the system improves, what caused it? The world getting richer, a lucky inference, the human approving better artefacts? It's hard to distinguish between these, and I don't fully understand why my system improved.

The human-as-approver model has a scaling problem. Nothing deploys until a human reviews it. This works at the scale of a UI testing framework. It isn't viable at the scale the AGI framing implies, because at that point the human approver becomes either a bottleneck or a rubber stamp. Automating the review with another AI just moves the trust question one layer up. This is an unsolved problem.

Self-modification deserves particular attention. Three things make it tractable here. The deterministic artefact layer means the AI experiments freely, but nothing deploys until a human approves; every modification is versioned and logged, so if a rewritten tool introduces a regression you can see exactly what changed and revert; and every conversation is logged, so you can see what the AI did and what it claimed its reasons were, even if those claims are

post-hoc rationalisation. None of this eliminates risk, but it makes the risk legible, and you can't manage what you can't see.

There are two claims here. The first is that building richer, persistent, self-modifiable environments around LLMs makes them more capable at practical tasks. The second is that this is a meaningful stepping stone toward AGI. I believe both, but with very different levels of confidence. The first carries its own weight regardless of how the second pans out, and the second is purely speculative. The practical payoff doesn't depend on the philosophy being right about intelligence; it just depends on the engineering being right about feedback loops.

The honest falsification. Whether this is just another magic incantation in the long history of LLM voodoo, I genuinely don't know. If a control run on the same project, with a clean repository and none of the persistent-environment discipline, produced comparable results, I would call this voodoo and move on. I haven't run that experiment. Anyone who does, please tell me what you find.

Despite these limitations, the systems I have built using this approach work materially better than the ones I built without it. And "built" is generous. I didn't write a single line of the automation code, the workflow definitions, or the MCP tooling. The AI did. I asked questions, called out speculation, reviewed artefacts, and approved results. That doesn't prove anything, but it's enough reason to keep going. This methodology won't produce AGI on its own, but it might be a useful step in that direction, and the practical improvements don't depend on the grander claim being right.

I built this for UI test automation, but I'm not a QA engineer. I'm a software engineer who ended up building a testing framework because the problem was there. A proper QA engineer would almost certainly do a better job with this methodology than I did. I have no idea how it applies outside of software engineering, to fields like legal research, clinical workflows, education, logistics, or anything else where AI is being wedged into processes designed for humans, but the core dynamic feels general. An AI that can modify its own tooling, accumulate knowledge in its environment, and be nudged by the world it inhabits should outperform one that starts fresh every time, regardless of the domain.

This paper was written with the assistance of Claude. The ideas, the experience described, and the arguments are the author's. The prose was developed collaboratively. The methodology described was developed in a commercial context; implementation details are omitted.

7 Related work

Several existing systems implement components of what this essay describes. Voyager [Wang et al., 2023] is an LLM-powered Minecraft agent that accumulates an ever-growing skill library and uses iterative prompting with environmental feedback. Generative Agents [Park et al., 2023] simulated a town of AI agents that accumulated experiences and developed distinct personas through reflection. Both exhibit features central to World Engineering: persistent state, environmental feedback loops, and behaviour that improves without fine-tuning.

Independently and concurrently, Andrej Karpathy's LLM Wiki [Karpathy, 2026], published as an idea file in April 2026, describes the same instinct applied to the knowledge layer. Raw sources are dumped into a directory, the LLM compiles them into a persistent, structured wiki, and linting passes maintain consistency over time. The human curates and reviews; the LLM authors and maintains. The pattern described in this essay is the operational-layer counterpart. Where Karpathy's wiki accumulates knowledge about a domain, World Engineering accumulates the agent's own substrate: tools, sensory apparatus, prompts, project rules, and the nudges embedded in tool responses. Both are instances of the same shift from per-inference curation to

persistent, agent-authored infrastructure. The convergence is itself evidence that this is a real direction rather than one person's framing.

What distinguishes the World Engineering framing from these prior and parallel works is bidirectional modification (the agent reshapes the environment itself), the deterministic artefact layer, and the design-for-the-inhabitant principle. These are differences of emphasis rather than kind, and someone could reasonably describe Voyager's skill library as world engineering avant la lettre. The contribution of this essay is the synthesis and the design philosophy, not any single component.

References

Anthropic Engineering. Effective context engineering for AI agents. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents, 2025. September 2025.

Andy Clark and David Chalmers. The extended mind. *Analysis*, 58(1):7–19, 1998.

Andrej Karpathy. LLM wiki. GitHub Gist, 2026. URL https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f. April 2026.

Tennison Liu and Mihaela van der Schaar. Position: Truly self-improving agents require intrinsic metacognitive learning. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *PMLR*, Vancouver, Canada, 2025.

F. John Odling-Smee, Kevin N. Laland, and Marcus W. Feldman. *Niche Construction: The Neglected Process in Evolution*. Princeton University Press, Princeton, 2003.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2023.