SUPERSOFTWARE

SOFTWARE THAT IS ITSELF SYMBOLIC AI

Authors:	Philip Sheldrake, Dirk Scheffler
Contributors:	Tudor Munteanu, Christina Wilpernig
Reviewers:	Rafael Kaufmann, Dave Lockie, Matthew Schutte, Tim Robinson

Published:	26 November 2025
DOI:	https://doi.org/10.5281/zenodo.17723706

Contact:	me@philipsheldrake.com

Table of contents

1. Executive Summary

The Problem. Trillions of dollars are flowing into LLM technology and services, and yet LLMs can't grapple with enterprise software.

AI-assisted software development is an awkward and risky alliance, as is any attempt to let LLMs act in the world through software. While they offer immense stochastic power, LLMs lack a verifiable ground truth and often violate architectural invariants, governance policies, security policies, and resource constraints. They need legible and coherent world models, yet they find a chaotic archipelago of ungrounded code bridged only by crude serialisation formats.

This isn't an LLM problem. It's a software problem.

Today's software paradigm emerged during the pre-networked era, to say nothing of the LLM age. '*The network is the computer*' remains more industry slogan than reality. While the current paradigm has been inadequate in its own right for years, its inability to meet the demands of LLMs let alone help realise their full value makes the problems all the more palpable. And costly.

The Solution. We introduce a new paradigm — software that is itself symbolic AI, poised to support intelligence *between* not merely *within*. Reactive and reflective symbolic systems are capable of inspecting and modifying their own form, and so help govern their own behavior. The simplified, behaviour-free model system brings significant advantages in terms of speed, resilience, adaptability, and security.

While others attempt to ease the symptoms, we're resolving the root cause. We call it supersoftware.

The Mechanism. We extend the principle of homoiconicity beyond code to the entire system. Application code, configuration files, data pipelines, API definitions, database schemas, dependency manifests, etc. are all represented in a reflective, symbolic structure, allowing the system to reason holistically about itself. Reasoning is enhanced the more supersoftware interacts with other supersoftware.

The Neurosymbolic Synergy. Supersoftware provides the missing half of the AI equation. The LLM provides intuition, intent, and context, and supersoftware provides a world model, grounding, validation, security, and architectural guarantees. LLMs generate accurate, explainable, and policy-compliant responses when guided and constrained by the supersoftware's verifiable logic. Supersoftware supports the requisite complexity with unparalleled simplicity — a true symbiosis.

The Vision. Inspired by biological processes, our approach moves software beyond mechanistic assembly towards an ecology of self-referential differentiation. We create systems that are computationally superior, by nature. This not only resolves many of today's engineering challenges but holds promise as a foundational component of Artificial General Intelligence (AGI).

2. Introduction

2.1 A new epoch

LLMs mark a new epoch of artificial intelligence and demand a new epoch of software. Software that is itself symbolic AI. Supersoftware.

LLMs are phenomenal. Even so, they have been championed in ways exceeding their innate capabilities. More than a handful of key opinion leaders have, effectively, claimed that LLMs will do it all. Everything.

Not to detract from LLMs' demonstrable value one bit, these soundbites are viewed by increasing numbers as hyperbole. In the words of the Gartner hype cycle, we have likely arrived at the peak of inflated expectations.

Either way, we don't have to argue the point but rather look to see how language models are being wielded. We find various ways in which LLMs are being partnered with symbolic data sources and what is somewhat affectionately called good old-fashioned AI. AI based on rules and logic. Symbolic AI.

IBM sees the hybrid approach – neurosymbolic AI – as a pathway to achieve artificial general intelligence.[1] Google DeepMind's AlphaGeometry solves geometry proofs by combining LLM-style generation with formal symbolic deduction engines.[2] Microsoft's Semantic Kernel brings a rules-based orchestration layer over LLM output.[3]

Supersoftware is the perfect complement to LLMs, playing to their astonishing strengths and making up for their stark weaknesses. A neurosymbolic synergy is formed, free from the encumbrances and inefficiencies of narrower intermediating techniques.

LLMs operate in the world through their direct users and through software. Our primary focus is software, yet this inevitably encompasses humans when recognised as cyborgs. Across the spectrum then — from Alice to global enterprise systems and big tech infrastructure — supersoftware will enable us to better govern the systematic effects of LLMs and help ensure outcomes that are more human-centred than they might otherwise be.

This matters all the more as we contemplate supersoftware's powerful contribution to artificial general intelligence.

2.2 The product

We define **supersoftware** as software that is itself symbolic AI.

Hiconic[4] is the foundational, enterprise-proven, open source homoiconic technology, stewarded as a public good. The brainchild of one of our co-founders.

Recognitive is being built up and out from Hiconic to make the technology more accessible, and for the purposes of creating and evolving supersoftware in partnership with LLMs. Recognitive is open source supersoftware helping everyone build supersoftware.

2.3 The technology

Technology is knowledge of how to fulfil certain human purposes in a specifiable and reproducible way.[5] It's not so much the artefacts per se but the knowledge that underlies the artefacts and the way they can be used. It's necessary then to include a rundown of the interdisciplinary knowledge we're calling on here in the last section of the paper — Our foundations.

It's the last section because readers rightly want to get to the heart of the matter — how we think about it , what it looks like in action, its commercial and societal value, and a fictitious case study illuminating the operational reality.

All hyperlinks in the main body of text are internal, often to information in the foundations section should any ideas be unfamiliar to you.

2.4 Who is this paper for?

The paper will appeal to AI experts, software architects and engineers, CTOs and CIOs. With the exception of the more technical aspects, it should also make sense to senior decision makers and policymakers who, by the nature of their role, have experience of the transformational effects of information technology.

3. How we think about it

3.1 Intelligence, cognition, and software

Wisdom begins with the definition of terms:

- **Intelligence** is the capacity for effective cognition
- **Cognition** is the process by which information is acquired, processed, and used to figure out how to act in the environment, i.e. sensing and knowing and acting.

While cognition is too frequently associated soley with brains, we adopt the more expansive perspective highly regarded amongst biologists:

> Living systems are cognitive systems, and living as a process is a process of cognition.[6]

To know then is to act effectively through structurally coupled interactions that sustain the organism's way of being, its ongoing viability.

By these definitions, all software has intelligence but only often at a most rudimentary level. Nevertheless, we give ourselves the advantage of considering a very long tail of intelligence rather than subjecting our concepts and their application to some arbitrary cut-off, and more importantly, we shift our framing here from computer science to the life sciences.

With this in mind, consider the following in our computing context:

- **Endo-intelligence** — inner, local, autonomous, self-in-relation
- **Exo-intelligence** — outer, network, collective, whole-in-relation.

Endo-intelligence is the bounded intelligence of data and code organisation.

Exo-intelligence is contingent upon the structural grammar that governs everything entering or leaving endo-intelligence. It thrives with the canonicalisation of form and form reflection.

Semantic normalisation is unnecessary and indeed undesirable. Meaning is contextual and plural by nature. It cannot be canonicalised without destroying necessary diversity and complexity — the Semantic Web's error. Canonicalisation here isn't mere normalisation but rather the act of bringing multiplicity into coherence without erasing difference.

Declarative grammar allows diverse semantic worlds to interoperate. Mediation layers, message buses, and persistence engines function without semantic awareness, a separation that makes exo-intelligence scalable.

Every process in a computer exhibits a certain endo-intelligence expressed through its internal structure — how it organises data in memory and connects code to data. Arrays, structs, classes, and objects form internal orders that determine how a process acts within itself.

Yet this intelligence is sealed.

Each programming language defines its own private cosmology of data and execution. Software are islands of reasoning left to communicate through crude serialisation formats — JSON, YAML, Protobuf, RDF — which flatten rich internal structures into impoverished syntax.

Such endo-intelligence lacks the most crucial faculty of higher intelligences — the ability to externalise its knowledge in a canonical, intelligible form.

Exo-intelligence begins where endo-intelligence ends — at the interface between processes. It's contingent on the way in which data, configurations, function calls, errors, and even executable behaviours are exposed, transmitted, and understood.

Disentangled, generalised exo-intelligence requires a grammar of cooperation — the structural logic through which intelligence can flow and evolve between processes that share no internal language. This in turn demands property containers with polymorphism support, graph structures with referential integrity, and a concise set of scalar base types.

We come then to supersoftware's emergent superpower.

Endo-intelligence and exo-intelligence co-evolve via recursive reflection and action.

When endo-intelligence is grounded in exo-intelligence, internal reasoning becomes clearer, safer, and more portable, and the boundary between computation and serialisation dissolves. Internal logic evolves in synchrony with its environment, with the world — a most natural feedback loop by which every process becomes both teacher and student.

Endo- and exo-intelligence are united through canonical reflective models, yielding raw performance, memory optimisation, interoperability, and conceptual clarity.

A new form of distributed, scalable, and efficient cognition emerges, and so then a new form of artificial intelligence.

3.2 Thinking about cognition

We can approach computing more usefully with the understanding that cognition requires an appreciation for how mind, body, and environment form an integrated, interactive system.

Cognitive scientists talk of 4E[7]:

- Embodied — cognition is shaped by and dependent on bodily processes
- Embedded — cognition is situated within and shaped by the broader social, cultural, and ecological environment
- Enactive — the mind and world are co-constructed by action and perception
- Extended — cognition is supplemented and enhanced by the environment, e.g. by our tools.

LLMs are qualitatively distinct from conventional software, which we'll refer to hereafter simply as software. LLMs operate over vast datasets and complex statistical relationships, enabling them to generate responses that appear fluent and adaptive, and giving the qualitative impression of cognition and knowing in the much stricter sense of having been subject to conscious reflection and intentionality.

Some researchers and philosophers argue that this can't then be genuine knowing, while others describe it as a form of functional or behavioural knowledge. As

nature selects for what works rather than how it's achieved, a focus on what LLMs do rather than how they do it suits our purposes here.

LLMs are bio-analogous, inspired by the design of the cortex, the part of our brain ideally suited to creativity. Software on the other hand is more analogous to the cerebellum, the system that takes that novel creation and ingrains it into the individual, enabling robust, repeatable, and flawless performance.

We call on the cortex to conceive new music and choreography for example, and rely on the cerebellum for the regular and precise performance of the song and dance.

To help push home the variety of natural mechanisms in play, imagine your finger lands on the hot stove as you complete that amazing dance move while cooking dinner. A reflex action kicks in for your protection. This rapid response is managed by the spinal cord allowing you to pull away in milliseconds without the delays of messaging and your brain having a think about it. Your brain does get updated a short while later and handles the perception of pain and the process of learning (perhaps less energetic disco moves in the kitchen in future). This whole process is an example of embodied cognition.[8]

The software-cerebellum analogy isn't commensurate because the disco dance and the reflex arc both entail a much richer structural coupling of the biological system with its environment than can be attributed to software's 'sensing and knowing and acting'.

Can software step up? Can we proxy for its lack of embodiment? Software's intelligence might be crude today, but tomorrow?

Nature tells us it could have more of a role to play, more value to offer. With this said, let's focus on a quality of software that isn't always front of mind.

3.3 World models

In living in the world, we all develop representations or simulations of it for reasoning and decision-making; so-called world models. It's well understood that LLMs don't 'do' world models — of chess or Othello[9] or orbital geometries[10], let alone anything approaching the complexity of social systems.

This assertion does pivot on what one considers to constitute a world model of course. A question of degree. One could argue that LLMs do form world models but inadequately, insufficiently, but that's not how we frame it. We think of LLMs in terms of "a bag of heuristics"[11] rather than world models; useful as we've all experienced, just not a world model.

This is a critical observation well worth dwelling on. As researchers from Berkeley, Stanford, MIT, Cornell, and Pennsylvania conclude[12]:

> Being able to effectively write code relies on having a strong semantic understanding (somewhat like a world model) of the entire codebase: structurally seeing how various parts of the code go together, knowing what is implemented where, understanding how the algorithms work, and keeping track of program invariants at certain program points. LLMs struggle with this global semantic understanding.

LLMs are fine during the opening stages of a game of chess because openings are very well documented. LLMs ingest these documents and use the consequent associations in their stochastic selection of the next move, but things begin to go very wrong once the opening game is concluded. The LLM doesn't have any appropriate intelligence to call on now.

The top-ranked chess engine is Stockfish. It's an example of symbolic AI often referred to as a classical engine, meaning it uses symbolic, rules-based methods combined with exhaustive search algorithms.

Another major contender is Leela Chess Zero, inspired by Google's AlphaZero. Both are neural networks, connectionist, conceptually similar to LLMs. Each learns the game entirely by reinforcement learning by playing many millions of games against itself. In other words, when there isn't sufficient information available to ingest, it creates the corpus itself.

Importantly, while self-play at scale may be possible in a non-physical realm determined by 64 squares, 32 pieces, and a rule book running to four dozen pages, your organisation is real and many magnitudes more complicated and complex. Obviously, playing millions of versions of your organisation's 'game' in advance of playing the real one is impossible.

However, this needn't be an either-or. Since 2020 for example, Stockfish integrates a neural network that works alongside the symbolic engine to assist in the evaluation of moves[13]. This neural network is trained on extensive analysis of positions and evaluations from the symbolic engine, as well as data from games played by top players. Two artificial intelligences and human intelligence are combined.

As LLMs cannot form world models of social systems, of complex adaptive systems, might software help out? Could it be the collaborator to offer up a world model?

Software is envisaged, architected, developed, and maintained with world models in mind. The engineering team will work hard to create and work to a common world model, and inevitably each and every one working on the software brings their own subjective and slightly varying mental models to the process. Importantly:

> Almost all the knowledge and information about the design decisions the architecture is based on are implicitly embedded in the architecture.[14]

Software is the digital representation of the world model(s) that informed its design and development. Unfortunately, it doesn't render the model(s) readily legible to itself, to other software, to its engineers, or in fact to anyone or anything. The legible map (model) and the territory are separate. The map is a static description of what the system should do, while the territory is what the system actually does at runtime. The application of LLMs to software engineering has had to explore routes around this poverty.

Could the system contain a model of itself that is legible and meaningful to itself and other intelligences with which it's in relationship?

Yes. Supersoftware does just that.

3.4 Reflection and homoiconicity

Self-referential systems are renowned in maths, physics, chemistry, biology, psychology, sociology, linguistics, and cognitive science. When subject to recursion and a system's ability to modify its own structure — reflection — they produce welcome complexity with the potential emergence of higher-order meaning,[15] and all the more powerfully when homoiconic. The biological archetype is foundational to all living processes.

> Living is information processing ... Life is both the data and the program ... there is no distinction between the 'machine' and the program — both are information.[16]

You might say that Mother Nature is giving us the biggest of big hints here. (See Bio-analogy and biomimetics.)

The roots of homoiconicity trace back to early formal logic, particularly Gödel numbering, which assigns unique integers to every symbol, formula, and proof within a formal system. Gödel numbering inspired fundamental ideas about self-reference and encoding that influenced computing's evolution, culminating in architectures where data and code can be treated interchangeably.

Yet few technology systems are homoiconic, frustrating many aspects of how our information technology operates and interoperates today. The Lisp programming language is a rare and marked exception, renowned of course for its profound and enduring relationship with the field of symbolic AI.[17]

While this was the dawn of universality, it came with a price. The architectural legacy of Gödelization yields flat, unstructured code. Meaning is contingent upon the executor's understanding — a number becomes an operation only because the interpreter expects it to be one.

Lisp for example represents a positionally structured homoiconicity. It's sequential, opaque, and limited to a single symbolic domain, i.e. an endo-intelligence. We refer to traditional homoiconicity then as endo-homoiconicity.

Our technology extends homoiconicity beyond language boundaries. Models are code and code is model, not through identical syntax but through shared canonical type-safe structure. It's exo-homoiconicity.

Every domain — configuration, invocation, error handling, business data, even behavioural contracts — becomes a first-class model in the same reflective ecosystem. Exo-homoiconicity transforms interoperability from tedious translation to intelligent structural canon. Whereas endo-homoiconicity unifies execution and representation, exo-homoiconicity also unifies communication and reflection.

As types and properties declare their own potentiality, algorithms can reason about form without domain knowledge or procedural context.

Our team members have been working on this broader and deeper appreciation of homoiconicity and reflection since 2010. The technology brings homoiconic, reflective, polymorphic modeling to any domain and provides the crucial technological basis for portable addressability of functions in distributed, decentralised, panlingual, and artificial intelligence environments.

3.5 Software and LLMs in collaboration

Are LLMs more able to 'work with' programming languages that are homoiconic versus nonhomoiconic?

There's a compelling argument supported by emerging research that homoiconic languages (e.g. Lisp, Scheme) could offer inherent advantages for LLM 'understanding' and code generation. Here are two recent perspectives.

> [Lisp's] hallmark feature — homoiconicity, or the uniform representation of code and data — naturally complements the capabilities of large language models, which benefit from easily parseable and manipulable structures. … Lisp's uniform parenthesized expressions reduce both cognitive and computational load — allowing language models to focus more on semantic content than on syntactic correctness.[18]

> [Lisp's] unique capacity to represent and manipulate symbolic information provides a distinct advantage for certain AI tasks. … LISP's core strengths in symbolic manipulation and metaprogramming uniquely position it to address the challenges of next-generation AI, such as the growing need for explainable AI and more sophisticated agent architectures. The potential for LISP to play a role in meta-programming for LLMs and AI agents presents a promising new direction for its application in the evolving AI landscape.[19]

LLMs are designed to master the statistical, semantically-rich, and structurally ambiguous world of natural language. When we apply these models to code, we hit a fundamental mismatch.

Lisp's endo-homoiconicity offers only flat syntax whereby meaning is opaque and contingent on the local context of an interpreter. While an LLM can learn to imitate the syntactic patterns of S-expressions, it cannot truly grasp their verifiable meaning because that meaning doesn't exist until runtime.

Recognitive's new exo-homoiconic foundation provides the explicit, verifiable ground that's been missing. The expansive self-referential structure helps form a denser and more interconnected web of syntactic patterns useful for token prediction than for nonhomoiconic languages or indeed for endo-homoiconic programming languages.

The LLM's role becomes translating ambiguous human intent into this precise structural grammar in collaboration with the supersoftware system, finally enabling a safe and powerful synergy between statistical intuition and logical reasoning.

In some ways, Recognitive might be considered Lisp's precocious grandchild — it's picked up a type-system and schemas down the family line, and disentangled exo-homoiconicity is the new trait. Unlike its grandparent, Recognitive imbues software with a deep, predictive understanding of its own structure. It can introspect and formally validate a proposed code change without running it, and reason about relationships, and yet its talents don't stop there.

Modern software systems involve far more than application code — e.g. configuration files, data pipelines, API definitions, database schemas, dependency manifests — and as noted in the previous sub-section, everything in a supersoftware system is homoiconic, acquiring a holistic reasoning capability in the process. This capability is most succinctly expressed by the following fictitious example of supersoftware's 'train of thought' articulated in partnership with a LLM:

> The LLM has proposed changing the signature of this function, and we've all agreed. My model shows this function is exposed via our public API. Therefore, this change is not just code; it is a breaking change to our API contract. We must plan to version the API, update the documentation, and notify our partners.

Supersoftware is a cognitive organism. It's the canonical world model of the organisation and its primary enactment, in constant collaboration with the organisation's other intelligences on both fronts. In light of these qualities, let's broaden and deepen the question at the top of this section.

Are LLMs better able to generate more informed, accurate, explainable, timely, and contextually rich responses when interacting with systems in which all domains are homoiconic and reflective compared with systems that don't have this quality?

Yes.

A paper co-authored by a connectionist pioneer found that the way a computer understands simple tasks and the way it understands how to learn new tasks have a lot in common, and that for such an application "our homoiconic approach significantly outperforms a nonhomoiconic baseline"[20]. This suggests that if an LLM could process coding problems and language constructs within such a unified, self-referential framework, its ability to understand code deeply and generate novel, correct solutions, especially for unfamiliar problems, will very likely be enhanced.

Supersoftware will allow LLMs to develop a more fundamental 'grasp' of programming logic, much deeper than surface-level pattern matching. An LLM interacting with a system where all domains share a unified, reflective, homoiconic symbolic system will have access to a profoundly richer and more integrated contextual ‘understanding’, leading to significantly superior responses across all dimensions.

Such combination transcends traditional retrieval-augmented generation (RAG), whereby an LLM might query a separate knowledge base or graph representation of code. It realises:

- **Introspection and status reporting**. The system can answer questions about its own current state (“What's the current configuration of service X?”), structure ("What are the dependencies of code module Y?"), or definitions (“What's the schema for data type Z?”)
- **Consistency checking**. It can reason about the internal consistency between its parts (“Is this API call signature compatible with the current API version definition?”, “Are all dependencies for this deployed code module met?”)
- **Impact analysis**. It can infer the potential consequences of a proposed change ("If I modify this part of the schema, which code modules, APIs, or data instances will be affected?")
- **Explanation of behaviour**. It can trace how a certain state was reached or why a component is structured a certain way by examining its models, rules, and history (“This API endpoint behaves this way because it's implemented by this code, which was governed by these configuration rules at the time, and processes data defined by that schema.”)

For informed and contextually rich responses, an LLM can retrieve not just isolated facts but the interconnections between, for example, a piece of data, its defining schema, the code that manipulates it, the configuration affecting that code, and the API that exposes it — all represented consistently. The supersoftware provides real-time information about its own state and structure, ensuring timely and relevant context and analysis. Accuracy is greatly enhanced because responses are grounded in this deeply interconnected, verifiable symbolic truth, where relationships are explicit not just inferred.

3.5.1 Enabling active inference

The core insight of active inference is profound, yet a primary challenge in its digital application lies in the nature of the model itself.

For an AI to engage in active inference, it cannot rely on static representation. The generative model must be constitutive of the agent, an operational substrate that can be acted upon and iteratively refined through the ‘lived’ coupling of agent and environment. Minimising free energy (a measure of the gap between what the

agent expects and what it actually encounters) thus demands a model that is dynamic and mutable from the outset; an innate quality of supersoftware.

3.6 Combining intelligences

Human intelligence has always included emotional, social, creative, and practical dimensions — intelligences with tens of thousands of years of history. What we now call systematic thinking emerged much later, codified during the Enlightenment into the dominant scientific and engineering paradigms. And while systems thinking was formally developed last century, its core intuitions have ancient roots in indigenous and philosophical traditions that were largely displaced by reductionism only to be revived as reductionism's severe shortcomings became more obvious to more and more people. This is a quick-fire way to reassert that all forms of intelligence have their role to play — human and artificial. Intelligence is always collaborative.

Table 1 outlines the role, value, and limits of each kind of intelligence.

Recognising their respective strengths and weaknesses is a first essential step to guiding their optimal combination. And then, per the insights of cybernetics, the combination and integration of intelligences demands recursive and reflexive interactions.[21] Or to put it more plainly …

Intelligences take turns influencing each other, maintaining and developing a meta-intelligence in the process.

3.6.1 Complexity with simplicity

This paper presents a new choice in the contexts of LLMs and software interacting: software can remain largely unreflective and comparatively dumb, or we can enliven it as symbolic AI to become an equal partner with LLMs. The latter is more straightforward than alternative ways to get LLMs and software to play nice. It's built-in, not bolted on, direct rather than intermediated.

Supersoftware supports the requisite complexity with much greater simplicity.

	ROLE	VALUE	BLINDSPOTS / LIMITS
Traditional scientific and engineering intelligence (systematic thinking)	Breaks problems into discrete parts, rigorously analyses and optimises them, ensures reliability, correctness, and efficiency.	Provides precision, quantitative rigor, and practical, tested solutions within well-defined boundaries.	Too often overlooks human, emergent, and relational dynamics.
Systems thinking intelligence (holistic / ecological thinking)	Making sense of things as interconnected wholes and relationships, with emphasis on feedback loops, emergent behaviours, and unintended consequences.	Guides strategic foresight, identifies systemic risks and opportunities, and helps avoid myopic optimisation that harms overall system health.	Can resist reduction of action if overly abstract.
Other human intelligences (emotional, social, creative, practical, etc.)	Brings context, intuition, cultural, ethical, and emotional insights and capabilities, and a diverse sense of values and priorities.	Anchor solutions in human realities, social dynamics, narratives, motivations, and ethical frameworks that pure engineering or AI may overlook.	Susceptible to bias, subjectivity.
Connectionist AI (LLMs, forms of neuro AI)	Processes and generates natural language, infers unstated context, abstracts across domains, supports creative problem exploration and dialogue.	Bridges human and machine communication gaps with conversational UI, surfaces latent insights from unstructured data, and augments human creativity.	Lacks world models, grounding, explainability, intent, values.
Symbolic AI (rule- and logic-based)	Encodes explicit domain knowledge, formal reasoning, and decision rules; can explain and justify decisions.	Enables trustworthy decision automation where rules are well known, compliance is critical, and transparency is required.	Potentially brittle in open-ended contexts.

Table 1: Categorisation and comparison of the intelligences discussed here.

3.6.2 Embodied and embedded cognition

Software today is deeply embedded in the organisation's environment, defining the landscape, the pathways, and the constraints in good part. But it's not a significant cognitive embodiment; more inanimate prosthesis than animate limb.

Supersoftware couldn't feel more different. It's an active, self-aware, reflective, and self-modifying part of the organisation's cognitive body, invoking biological analogies such as feeling and healing. Supersoftware can communicate its internal, reasoned state to LLMs directly, symbolically, and to us humans symbolically and in natural language via a LLM.

Supersoftware co-evolves as and with the organising, as and with the organisation's economic, social, technological, and environmental contexts.

3.6.3 Extended cognition

To say that software is a tool and supersoftware is an AI tool is to miss the fecundity of the interweave of intelligences here. Supersoftware is a new form of collaborative extended cognition. It's the difference between a tool that extends your memory — e.g. a CRM system — and a collaborative partner that extends your capacity for reason.

3.6.4 Enactive cognition

Software enacts business processes deterministically and rigidly. Supersoftware encompasses the co-construction of knowing and the world through the continuous loop of perceiving the organisational environment and acting upon its own code.

Imagine the collaboration of intelligences of which supersoftware is part perceiving a change in its world, perhaps a new security threat or regulatory requirement (see the Fictitious case study). In such collaboration, supersoftware modifies its own structure and so simultaneously co-constructs its own symbolic 'understanding' and its world.

3.6.5 World model and meaning

The vast majority of organising entails software. With supersoftware, the corresponding world model *is* the software in good part, made dynamic, real-time, and legible to all intelligences in the mix. Reflecting it back at them. Similarly, supersoftware is the territory and often the medium by which decisions reached by the combined intelligences are implemented.

We can consider four parties in the organising: software engineers, everyone else, software, and LLMs.

With supersoftware in the mix, all four parties are palpably intelligent and their combination all the more so. All four parties 'speak the same language' with shared meaning in the models. All four parties can discuss the models with the necessary subjectivity and contextual relevance, supported by the precision of the underlying model grammar.

The shared 'language' allows software itself to validate that any proposed change adheres to its core, provable structure. Intent is made a living, computable part of the system.

What were once merely external prompts become active and formally modeled constraints that directly guide and bound the AI's scope.

3.6.6 Governance and control

The LLM is no longer an untrusted external tool in the contexts of software design and development, but an integrated creative partner. The combination of LLM and supersoftware is a powerful implementation of neurosymbolic AI. The readiness with which the system can ascertain what should be, what is, and how best to close the gap meaningfully and accountably, transforms the potency and cogency of the associated governance processes.

Being built-in rather than bolted on, being direct rather than intermediated, the system is self-aware and capable of far greater self-control.

3.7 AGI

AGI is characterised by a fluid intelligence able to rise up to challenges it hasn't seen before.[22] The value of Recognitive doesn't pivot on it making a contribution to AGI. Nevertheless, it does.

3.7.1 By evolution

With mass adoption of Recognitive and innate to its biomimetic qualities, we expect the very natural emergence of new architectural structures in just the same way biological systems evolved new cell types, organs, and nervous systems to achieve higher levels of complexity; albeit over a significantly compressed timeline.

As degrees of reflection increase and structures proliferate, LLM model size reduces significantly and next-level capabilities materialise, i.e. a potential new form of artificial intelligence emerges, plural and decentralised by nature.

3.7.2 The grounding problem

A major barrier to AGI is LLMs' inability to reason reliably or ground their outputs in a verifiable model of reality — the grounding problem.[23] AI systems process patterns of symbols without grounding them in lived perception or embodied reality. True general intelligence very likely requires embodiment, i.e. the ability to interact with and learn from a real environment.

Supersoftware grounds LLMs with the world through us and our 'things'[24] (see World models).

3.7.3 Intermediary strata

There isn't a simple jump from neural firing patterns to explicit symbolic thought in biological systems. There are intermediary strata — embodied, social, and ecological dynamics.

Nevertheless, the dominant mode of neurosymbolic AI research, specifically the elicitation of symbolic knowledge from neural networks in pursuit of tight neurosymbolic integration, assumes such strata can be abstracted away. While this is wholly aligned with computer science methodology and has the advantage of being more readily formalised, and while it may also best serve narrow pragmatic goals, it's sub-optimal in the context of combining human and artificial intelligences naturally and powerfully.

Recognitive addresses the intermediary strata between raw neural processing and formal logic. That stratum is us, effecting a tight and grounded cognitive loop with the technological system.

This creates a powerful feedback loop best described by the concept of enactive cognition. Rather than the AI simply being a tool we command, we have our minds, the world, supersoftware, and LLMs co-constructing one another through cycles of action and perception.

3.7.4 Program synthesis — a foundational component of a form of AGI

The vista here is captured well on the website of new startup[25] founded by a leading AI expert:

> The path to AGI is not through incremental improvements to existing methods. The problems with deep learning are fundamental and cannot be addressed superficially. It's time for a new paradigm. The good news is that we know what it is: program synthesis.
>
> … Instead of interpolating between data points in a continuous embedding space, program synthesis searches for discrete programs, or models, that perfectly explain observed data. This allows it to achieve much greater generalization power with extreme data-efficiency, requiring only a few examples to learn. … We believe program synthesis and deep learning are equally important.

Program synthesis is a class of techniques able to generate a program from a collection of artefacts that establish semantic and syntactic requirements for the generated code. Specifically in AGI contexts, it's the process by which an intelligent system builds explicit, reusable programs to solve new problems from minimal data rather than relying on pattern-matching from massive datasets.

As an advanced program synthesizer, Recognitive is a foundational and catalytic component of this form of AGI. LLMs ingest and generate supersoftware with a far deeper and more useful 'grasp' of its intent than is possible with a corpus of conventional code, and supersoftware can be instrumented and evolved dynamically without ever needing to be recompiled or redeployed.

Recognitive aligns perfectly with this 'new paradigm' of program synthesis. We share the vision with a significant qualifier: any program synthesis is very much more likely to serve the pursuit of AGI as and when all domains share a homoiconic, reactive and reflective symbolic system.

More than a concrete embodiment of the vision, Recognitive channels the deep natural processes that inform the evolution of cognition. The paradigm may be new in computer science but is otherwise as old as life itself.

4. What it looks like in action

4.1 Meaningful modeling

We apply homoiconic data modeling to strongly decouple data from semantic and technical domains, e.g. persistence, exchange formats, protocols, and event-sourcing. Normalised models are the pivotal interface for data, configuration, functionality, and errors, and all models are extricated from technological aspects such as processing code or programming languages, consciously breaking with OOP (object-oriented programming) encapsulation in favor of expert late binding, as we find in nature (see Genetics).

Models can be forked, shared, and merged. Vitally, in light of our biomimicry, such diversity of contextual modeling affords variation, selection, inheritance, and speciation, which together constitute the neo-Darwinian understanding of evolution. The models are analogous to the organismic genotype.

Such an approach begins to move us on from the struggles of the semantic web to the realisation of the meaningful web. Semantic here refers to the precise definition of words and their relationships within a specific language system; the objective technical aspects. Meaningful relates to the overall significance and implications of something, often involving subjective (i.e. human) and contextual interpretation.

You might say the semantic web is technical and totalizing whereas the meaningful web is contextual, plural, natural, human.

Subjectivity and contexts are celebrated for both domain agnostic and domain specific applications, yet a strict dedication to reflectivity maintains model inter-relatability.

4.2 Domain-agnostic algorithms

A new kind of computation emerges when models and their instances are canonicalised and reflectable — the domain-agnostic algorithm.

With structure made explicit, domain-agnostic algorithms are no longer limited to elementary tasks such as copying or comparing. They can traverse, validate, transform, merge, patch, serialise, replay, or persist any data form, guided purely by its declared schema. They can analyse relationships, track causality, and preserve referential integrity, all without semantic awareness.

They operate on the reflective declarations of type and property only, enabling a shared infrastructure of algorithms that serve every possible domain without specialisation:

- Serialisation becomes intrinsic
- Transmission decouples from implementation
- Versioning, patching, and replaying become structural operations.

Exo-homoiconicity turns reflection from a passive mirror into an active instrument of computation, of cognition, of intelligence.

An example domain-agnostic algorithm is included in Appendix 1.

4.3 Transforming software engineering

4.3.1 Reducing entropy

The core challenge in software engineering is identifying patterns and abstracting them into reusable routines to reduce entropy. Nevertheless, traditional high-level languages impose structural complexity that hinders this process, especially when abstraction goes beyond domain-specific logic.

Reflection in Java for example attempts to offer flexibility but remains inefficient due to the arbitrary diverse nature of class structures — methods, parameters, return types, static and instance members, and more.

Reflection becomes trivial and entropy is reduced when entities follow a strict structure — typed properties that are primitives, collections, or references to other entities.

4.3.2 Networked services

The reduction to enumerating properties eliminates unnecessary complexity.

When every entity follows a well-defined schema, there is no need for verbose conversion layers or complex adaptation logic. By minimising entropy, the system remains resilient, extendable, and easier to reason about.

The simplicity of this paradigm makes it particularly powerful in networked applications and distributed software architectures where state synchronisation, data marshaling, and distributed processing require low-entropy, predictable structures.

4.3.3 Structural simplification as acceleration

By limiting models to properties with types, reflection becomes trivial, native, and fast. There are no overloaded methods, annotations, or opaque policies, only typed readable and writable properties. Reflection ceases to be a costly meta-operation and becomes the natural interface to data.

4.3.4 Memory layout and access speed

The simplicity of canonical models allows optimised, cache-friendly memory layouts. Property access and interception become predictable, enabling high-speed cross-cutting operations such as change recording and dependency tracking.

4.3.5 Textual declaration as resilient representation

Being purely declarative, models can exist as plain text — lists of property and type names. No binary formats. No compiler dependencies. The system is transparent, correctable, and self-healing. Schemas can be inspected, diffed, and reloaded without downtime or data loss.

4.3.6 Runtime evolution and learning

Exo-homoiconicity enables runtime type addition and discovery. A running system can safely learn about new model schemas and use them type-safely, enabling live evolution without redeployment.

4.3.7 Balancing weak and strong typing

Exo-homoiconicity allows instantiation with or without complete schema knowledge, enabling flexible, error-tolerant behaviour. This hybrid capability fuses the safety of static systems with the adaptability of dynamic ones. Canonical structure replaces binary complexity with elegance, clarity, and unrivalled performance.

4.3.8 Reflection and native compilation

Recent trends in language and runtime design have shied away from code reflection, considering it incompatible with tree-shaking and native compilation. Frameworks such as GraalVM illustrate this tension. Reflection is seen as an obstacle to compact executables and deterministic optimisation.

Our technology secures the best of both worlds. Only the canonicalised models remain reflectable, the expert code does not — again, just as we find in nature. If behavioural code wishes to expose itself to reflection, it can do so intentionally through an explicit instruction model or annotation.

Selective reflection sustains and nurtures exo-intelligence while allowing internal logic to remain closed and undergo compile time and runtime optimisation such as tree-shaking, inlining, obfuscation, and jitting. Since canonical model reflection deals only with property and type declarations — not instruction trees or runtime dispatch — any impact on size and cost is negligible.

4.3.9 Non-von Neumann

Non-von Neumann ideas can overcome the integration challenges of significant architectural heterogeneity, data representation, communication, and instructions execution.[26] A post-von Neumann network-based computing architecture doesn't just become possible but likely.

4.4 What about ...?

Table 2 compares our well-defined and reflective model type system with GraphQL, Hibernate, Protobuf, message buffers, and JSON schema.

FEATURE	HICONIC / RECOGNITIVE	GRAPHQL	HIBERNATE	PROTOBUF	MESSAGE BUFFERS	JSON SCHEMA
Type Reflection	Yes	Yes	Yes	Limited	Limited	Yes
Instance Reflection	Yes	No	Convoluted	Limited	Limited	No
Inheritance (polymorphism)	Multiple	Single	Single	None	None	None
Technical Domain	Agnostic *	Graph description	Persistence	Serialisation	Serialisation	Serialisation
Property Access Cross Cutting	Yes	No	Convoluted	No	No	No
Metadata	Yes, modelled, agnostic	Limited	Convoluted	Limited	No	Limited
Structural Normalisation	Modelled axiomatic homoiconicity	Heteroiconicity	Heteroiconicity	Heteroiconicity	Heteroiconicity	Representational homoiconicity
Primitive Type Quality	Precise	Underspecified	Precise	Underspecified	Underspecified	Highly underspecified
Generic Data Operation Richness	High and growing	None	Framework-local	Framework-local	Framework-local	Framework-local
Isomorphic Translation	Yes	No	No	No	No	No

Table 2: Comparison table.

* i.e. incorporates all of the above mentioned technical domains and is open to support any specific domain.

4.5 AI-augmented software engineering

While LLMs promise a revolutionary leap in the velocity of software development, the promise remains largely beyond reach, all the more so for organisations operating in high-assurance environments.

Generative AI's lack of understanding of architectural integrity, and its inability to form and maintain a world model of the enterprise and its software, make it unsafe.

LLMs produce technical debt and violations of the system's fundamental architectural invariants, governance policies, security policies, and resource constraints at unprecedented rates. Despite claims of 2x, 5x, and even 10x productivity gains, it appears engineering productivity actually declines[27], although it should also be said that the subject remains contentious.[28]

Various attempts are being made to resolve some of these problems, as outlined in Table 3.

Further to the blindspots / limits listed for each, it's noteworthy that they are all interventions. Intermediations. Translations. Complications. Some are more probabilistic than engineered assuredness. All assume software remains comparatively dumb.

Few engineering teams, trending to zero in high-assurance software contexts, would be content leaving an LLM to much at all.

Reliance on LLMs for maintaining and developing software produces an awkward and risky alliance falling far short of the desired and indeed required symbiosis.

Recognitive doesn't just address these problems, it transforms the paradigm of software development.

	ROLE	VALUE	BLINDSPOTS / LIMITS	EXAMPLES
Graphing Techniques	Uses graph-based representations to model relationships between software components and their interactions.	Enhances understanding of dependencies and relationships in software architecture.	Complexity in graph construction; may require extensive data.	CAST, Apiiro, Neo4j
Scripting, Automation, and Orchestration	Enables custom scripts that allow LLMs to interact with development environments and tools.	Streamlines workflows and automates repetitive tasks, improving efficiency in development.	May require technical expertise; limited to the scope offered by each tool. Only has secondhand knowledge of the software.	Tessl, GitHub Actions, Microsoft Semantic Kernel, n8n
Fine-Tuning and Transfer Learning	Adapts LLMs to specific programming languages or frameworks through targeted training on relevant codebases.	Improves accuracy and relevance of code suggestions and documentation generation.	Requires substantial domain-specific data; risk of overfitting.	OpenAI Codex, Hugging Face Transformers
Prompt Engineering	Crafts specific prompts to guide LLMs in generating relevant code snippets or documentation.	Enhances the quality of code generation and documentation, making interactions more effective.	Effectiveness can vary; may require iterative testing to refine prompts.	PromptBase, Replit
Reinforcement Learning from Human Feedback (RLHF)	Uses human feedback to train LLMs on coding practices and software development standards.	Improves alignment with developer intentions, leading to more satisfactory code suggestions.	Dependence on quality of feedback; may not generalise well across contexts.	OpenAI's ChatGPT, Anthropic's Claude

Table 3: Comparing approaches to improving LLM ‘understanding’ of software and assisting their contribution in developing and maintaining software.

4.6 Grammar

The grammar is a convenient syntax for text-based formal model declaration. It's used to declare the model identity, model dependencies, and model types with their properties and constants. Metadata may be added to the model's structural elements (model, type, property, constant), both eagerly and lately when extending a model. The grammar can declare any model, and starts with the model that represents the grammar itself.

The grammar necessarily describes any data graph of any model, making it expressive in terms of model structure and generic when it comes to describing complex metadata instances. Data types can have intricate deep structure.

All expressive syntax elements allow continuation on a generic level (like the metadata) in order to keep the format open beyond its well-known structures.

```
entity Greet extends ServiceRequest {
String name

String eval()
}
```

4.7 IDE

The IDE is an application or plugin to existing IDEs for model designing, either with the grammar or graphically.

The IDE helps with syntax colouring, AI-assisted code completion, hyperlinks to follow the referential identifiers in a model to its declarations, and modularisation. The graphical design helps with convenient navigation and extension of the graph describing the model.

The IDE helps to refactor models in various ways. It dynamically resolves model dependencies and is able to pull / push designed models to repositories.

With the advent of natural language programming, our IDE will include an AI agent available to the AI agents of all variety of IDEs, facilitating the masterful application of our technology in those environments.

4.8 Repository

The repository stores models in their native, grammar-defined form and makes them dependable for the IDE or concrete programming languages. It has ‘faces’ to view the dependable deliverables in a language specific form such as NPM, Maven artefacts, Ruby GEMs, etc. And a ‘face’ designed for human and artificial intelligences with rating, description, and comment metadata.

4.9 Panlingual exchange format

In order to store and transport type-safe model data referentially, comparably, polymorphically, and efficiently, and so it may be deep nested and streamable, we offer an upgrade from JSON, YAML, and XML with a scalable exchange format supporting:

- Proper base types with their efficient literal syntax:
  - string, boolean
  - various number formats instead of just number
  - time types (data, time, datetime, etc)
  - binary data
- Native references — backward and forward
- Includes
- Placeholders
- Type inference, type explication
- Reference pools of values to maintain a flat syntactical structure, even when representing potentially deep, complex data graphs (avoiding indentation hell).

Where a panlingual exchange format helps:

- Data integration — reducing the ‘integration tax’ that consumes up to 50% of IT budgets
- Multi-cloud and multi-domain consistency — eliminating need for custom normalisation across cloud providers and services across the overall system
- Interoperability — enabling better tool integration and data sharing.

By way of a comparison, Google’s Protocol Buffers allows data and function structures to be serialised in a domain-bound way that optimises them for network transmission across diverse computing environments. Recognitive achieves much

more. Our polymorphic domain-agnostic models assist transmission but also function, persistence, reflection, metadata, presentation, configuration, and errors.

The panlingual exchange format will be considered a significant upgrade from model context protocol.[29]

ETL pipelines are managed by data teams, which are often siloed under a Head or VP of Data and include their own dedicated engineers and product managers. Even when these teams operate efficiently, the rest of the company must integrate with or around their work.

Traditional ETL processes follow static, predefined rules: data engineers write code to transform data from point A to point B, while data analysts interpret what those engineers have implemented to understand what stakeholders want to know.

As supersoftware treats transformation logic itself as data:

- Rules can be dynamically modified and composed
- The same framework that processes your business data also processes the transformation rules
- Operations become auditable, portable, and replayable across your entire stack
- You get automatic format manifestation rather than manual pipeline coding.

The integration tax is eliminated because transformations become self-describing and composable rather than hardcoded pipelines.

4.10 Programming languages

It's worth stating up front here that adoption of Recognitive is far from contingent on a native programming language, which is just as well given that language adoption takes many years. Rather, Recognitive is initially embedded in developers' familiar environments, e.g. Java, TypeScript, Python, Rust, Go.

4.10.1 Towards exo-intelligent programming languages

Eventually, languages that wish to thrive in a disentangled, generalised exo-intelligent ecosystem will embed canonicalisation as a first-class concept. They will provide:

- A universal reflective core based on canonical models
- Efficient traversal and mutation for domain-agnostic algorithms
- Native property containers and referential graph support.

Such languages can vary in style, syntax, and/or philosophy, yet remain interoperable through shared structure. This is not another framework or protocol, but rather a cognitive meta-layer. It's a way for all languages and runtimes to understand their own forms in a common mirror.

4.10.2 A native programming language

A future native programming language is as specified as possible based on the model system. In other words, models define the programming language and keep it normalised for common processing by all variety of tools. The language's essential models include:

- instruction-model
- module-model
- package-model (to identify deliverables and their dependencies).

The programming language has expressive grammars for the most important models, and strong literals to reference its own structures for reflective, generic programming (self-instrumentation):

- this method
- this class
- this instruction
- this module
- this field.

The programming language handles dependencies automatically to avoid version clashes. It distinguishes between dependencies for module-internal requirements and dependencies for module-external sharing.

How should it stack up?

Table 4 portrays Lisp's comparative strengths in the context of AI programming, reproduced from Quantum Zeitgeist magazine[30] with the addition here of a column for the Recognitive language.

FEATURE	LISP	PYTHON	PROLOG	JAVA	RECOGNITIVE
Paradigm	Functional, Procedural, Object-Oriented	Multi-paradigm (Imperative, Object-Oriented, Functional)	Logic Programming	Object-Oriented, Imperative	Modeled, Homoiconic, Polymorph,Functional (Expert), OOP
Syntax	Parenthesised (S-expressions)	Indentation-based	Rule-based	C-style	Property Access, Flexible Object Literals (wiring)
Typing	Dynamic	Dynamic	Dynamic	Static	Primarily static
Memory management	Automatic (Garbage Collection)	Automatic (Garbage Collection)	Automatic (Backtracking)	Automatic (Garbage Collection)	Automatic (Garbage Collection)
Meta-programming	Powerful macro system, Homoiconicity	Limited	Limited	Reflection	Model / Data Reflection, Homoiconic self-instrumentation
Community & libraries	Smaller, focused on symbolic AI	Large, extensive libraries for various AI tasks	Smaller, strong in logic programming	Large, libraries for general AI and big data	Holistic and versatile libraries for domain agnostic model operations
Performance	Generally faster	Generally slower	Can be efficient for logic-based problems	Robust performance	Robust performance + reflection boost for domain agnostic operations
Learning curve	Steeper	Easier	Moderate, requires a different way of thinking	Moderate	Moderate
Symbolic reasoning	Excellent	Limited	Excellent	Moderate	Excellent regarding all homoiconic modeled features

FEATURE	LISP	PYTHON	PROLOG	JAVA	RECOGNITIVE
Machine learning	Historically used, integration efforts ongoing	Dominant, extensive libraries	Less common	Growing with libraries like Deeplearning4j	Well-suited via normalised models and homoiconicity
NLP	Strong historical presence	Widely used with libraries like NLTK and spaCy	Well-suited for parsing and understanding	Used, but less dominant than Python	Well-suited via normalised models and homoiconicity
Expert systems	Historically significant, still used	Possible, but less natural	Well-suited	Possible	Well-suited via its normalised models and expert bindings. Homoiconicity and expert systems go hand in hand.
Scalability	Can be scalable, Clojure on JVM offers good concurrency	Scalable with appropriate frameworks	Depends on the implementation	Designed for scalability *	Designed for scalability

Table 4: Portraying Lisp's comparative strengths in the context of AI programming, reproduced from Quantum Zeitgeist magazine with the addition here of a column for the Recognitive language.

* We don't consider Java well suited for scalability in distributed computing contexts because, unlike Recognitive, none of these languages really care for 'data on the move'.

5. Better

How will Recognitive improve things? For example, how might it contribute to human flourishing and to our abilities to live in harmony with all other life on Earth? More pragmatically and imperative in terms of adoption, what's the business value? As the latter affords us this opportunity, we'll start there.

5.1 Business value

Organisations exist to create value that could not be created otherwise. Value for customers. For shareholders. For employees. Partners. Suppliers. Citizens. Any lingering idea that an organisation is defined by its payroll is relegated to the 20th Century where it belongs. An organisation is reliant upon all these participants to sustain and grow its ability to create value, and if the participants do not find the value they're looking for, they'll seek it someplace else.

To avoid invoking hard boundaries, we find that thinking in terms of organising rather than organisation better reflects this reality.

Definitionally, organising involves cognition and intelligence: the means and capacities by which your organisation acquires information, processes it, learns, and uses it to figure out how to act in its market. While it might be expressed in all kinds of ways, the ultimate question in every boardroom, at the heart of every strategy review, and permeating every management conversation, is:

> How might we organise better to create more value today and tomorrow?

For clarity, per living systems theory, we take organising to include being inventive and experimental, responding creatively when circumstances inspire and indeed require.

5.1.1 Superorganising

IT-business alignment is the degree to which business and IT depend on one another and share their domain knowledge to achieve a common goal.[31]

Recognitive has immediate value in the IT context, as should be apparent to any reader of this paper, yet its deeper transformational value demands structural and cultural adaptation. The challenge is so stark that much research into the topic presumes IT and the business to be diverse and separate organisational areas[32], which is far from a good start.

Business leaders have to involve IT leaders. Business and IT professionals have to collaborate in defining problems, in setting goals, and in determining what to do. Recognitive is the enabler.

The rewards are substantial in terms of agility, relevance, effectiveness, cooperation, competitiveness, and overall business value.

We refer to the diligent integration of human and artificial intelligences in pursuit of the purpose(s) at hand — an integration contingent upon supersoftware in light of the limitations of LLMs — as superorganising.

5.2 Social impact

This topic is of course much bigger than one section of a white paper but it's important to at least point in the direction of a more extensive future exploration.

Technology is never neutral. Its conception is always grounded in a set of contexts, worldviews, values, and ends in mind. Technologies encode practices and values into the societies that adopt them.[33,34]

5.2.1 Our design values

Value Sensitive Design[35] (VSD) engages "human values in the design of tools and technology to support human flourishing". A value is "what is important to people in their lives, with a focus on ethics and morality." It considers individual, group, and societal levels of analysis and observes that "in the complexity of human relations, values sit in a delicate balance with each other. This framing positions designers and researchers to emphasize moral and ethical values, but to do so within the complexity of social life, and with recognition for how culture and context implicate people's understanding and experience of benefits as well as harms and injustice."

The Recognitive team holds these values:

- We value digital systems that can be shaped by everyone, individually and collectively

- We value digital technologies that assist communities in their implicit and explicit sense-making
- We celebrate and nurture what is essentially human over the essentially technological, yet fully embrace technology's potential to promote human flourishing.

For example, taking each value in turn:

- With Recognitive's emphasis on readily understood symbolic models and a conversational interface, all stakeholders can engage directly with and in the system rather than via detached representations of the system of varying veracity
- The platform's symbolic reasoning core is designed to surface logical inconsistencies and hidden assumptions, providing a concrete foundation for collective deliberation
- Human intelligence is foregrounded when it should be.

If your idea of better coincides with these values, then the conception, design, and development of Recognitive is off to a good start. But what about supersoftware developed with Recognitive? This has to be open to VSD by definition. Let's take a quick tour of accountability, decentralisation, cooperation, and the vision of social cyborgs.

5.2.2 Accountability

In light of its inordinate advantages, supersoftware will eventually supplant conventional software. How then might its use be subject to societal governance? i.e. nudging supersoftware-based systems towards supporting if not in fact manifesting societal good.

Accountability (see On governance) is a core concern to the cooperative talents of the human species. It's a natural process albeit one that demands scaling with scale. Supersoftware elevates governance and enables metagovernance, i.e. emphasising coordination and cooperation between different systems of governance, making values and effects legible, and empowering community participation in the process.

With the benefit of determinism, ethical guardrails and failure modes can be modeled and made widely available, and their adoption made an auditable quantity. The Fictitious case study portrays verifiable compliance with a new regulatory requirement, and verifiability applies whether the system property in question is a legal requirement or any other kind.

5.2.3 Decentralising

Concentrations of power must be avoided whenever possible. Yet of course LLM training is a specialist undertaking demanding very deep pockets, economics that inevitably concentrates power.

By contrast, the cost of developing supersoftware and the cost of configuring the use of supersoftware should be less than for conventional software today making it accessible to all and infinitely customisable. Supersoftware keeps LLMs and so LLM vendors in check in decentralised systematic contexts. While it does not address the problematic power concentration directly, it curtails potential abuses.

5.2.4 Social cyborgs

No-one reading this can be described or understood solely in terms of the biological, psychological, and social. You are bound together with information technology. Your sensing is digitally augmented. Your sense-making and cognition are digitally augmented. Your acting in the world is digitally augmented. These are definitionally essential living processes. You are then cyborg.

There's no doubt we can massively improve and cohere and grow our cyborginess, not least by a dedication to our innate social qualities. We refer then to social cyborg, and that's social in the true sense of the word — i.e. the nature of human interaction — rather than the way in which the word has been co-opted in web2.

Supersoftware will help realise social cyborgs with the autopoietic concept of organisational closure. This is compatible with the hopes of those championing so-called 'self-sovereignty' without the contradictions and unnatural implications that idea brings with it – autonomy in nature is always relational. A topic for a future paper.

5.3 Cooperation at scale

Cooperation may be considered within business value, and within social impact, and also as a connector between these two realms as if perhaps they are indeed one and the same.

With unprecedented interoperability (see Panlingual exchange format), supersoftware enables a collective intelligence of supersoftware-based systems. We can consider a 'hyperorganising' including the integration of non-human, non-artificial intelligences and / or proxies.

6. Fictitious case study

This fictitious case study provides a vignette on the early operational reality of supersoftware which can get lost in a paper describing the big picture vision.

6.1 Introduction: agility vs. assurance

Adapt Financial is a leading financial services institution with over 10,000 employees and a global technology team of 800 engineers.

The company's core trading platform, Odyssey, is the integration of supersoftware (i.e. developed with Recognitive) and the company's LLM of record. (This case study will now refer to Recognitive / supersoftware and the LLM accordingly rather than Odyssey as an integrated whole.)

The technological architecture and approach is central to the company's strategy of maintaining elite performance and high-assurance security in a hyper-competitive, highly regulated environment. Errors can cost millions and yet responsiveness is a fundamental competitive edge.

A new regulatory mandate was issued in Q1 2028, Directive 2095-08B, requiring all institutions to log and report transaction finality latency for a specific class of over-the-counter (OTC) derivatives. The directive came with a tight deadline of 30 days.

6.2 Understanding: from prompt to formal constraint

6.2.1 Formalising the objective

Instead of a human team beginning weeks of manual code archaeology, Adapt's lead architect presented the text of Directive 2095-08B directly.

The text is not treated as a simple prompt for a generative model. Rather, the neurosymbolic system (Recognitive + LLM) parsed the regulatory language and translated it into a new, formally modeled constraint on its own behavior. The requirement to "log and report transaction finality latency" became a verifiable constraint added to its existing set of operational goals and constraints.

This act turned a piece of external text into an intrinsic, bounded problem space guiding all subsequent actions.

6.2.2 Contextual analysis

Recognitive queried its own structure, identifying the key components involved: the `Executor` module and the `ChronoBus` message bus.

It passed this context — the new formal objective combined with its own architectural model — to the integrated LLM with a clear instruction:

> Propose modifications to satisfy this new objective.

6.3 Planning: immune system and impact analysis

6.3.1 Plan A

- The LLM proposed an initial course of action: a simple logging approach.
- Working to each other's strengths, the Recognitive + LLM combination concludes that this is a naive approach. Attaching a synchronous logger to `Executor` would introduce a variable 5-8 millisecond latency during peak trading volume, violating a core platform SLO.
- Rejected.

The core symbolic logic acts as a biological immune system, rejecting the idea deterministically without a human needing to be in the loop. Nevertheless, the Lead Architect opted into the train of thought:

> PROPOSAL REJECTED. Violation of Architectural Invariant #A-14: 'No direct, synchronous messages emitted by the core execution path, to avoid latency at scale.' The proposal constitutes an unacceptable performance degradation.

6.3.2 Plan B

- The LLM proposed creating an endpoint on the `Executor` for another service to query.
- Recognitive finds no violations.
- Recognitive informs the LLM of the technical implications deterministically.
- Courtesy of contextual cost information available to the LLM via RAG, the LLM concludes this would significantly increase infrastructure costs and complexity.
- Rejected.

6.3.3 Plan C

- The LLM proposed modifying the data packet on the `ChronoBus` to include a high-precision `origination_timestamp` at the point of creation. And a lightweight, asynchronous `LatencyObserver` microservice subscribes to the `ChronoBus` feed and calculates finality latency without ever touching the critical execution path.
- ✓ OK. Continue.

6.3.4 Plan C is progressed for automated impact analysis

- Ephemeral Profiling: Recognitive spun up a sandboxed, in-memory clone of the `Executor` and `ChronoBus` modules with the proposed code changes applied.
- Telemetry-Driven Simulation: Leveraging existing telemetry, Recognitive ran a high-fidelity simulation, replaying data from the previous day's peak trading session to profile the performance of the modified code with zero human intervention.
- LLM-Generated Report: The structured simulation results — latency histograms, CPU / memory footprints, and security scan outputs — were passed to the LLM for synthesis into a human-readable impact report:

  Predicted impact on core `Executor` latency: <0.1ms. New `LatencyObserver` resource footprint: minimal. Security posture: `LatencyObserver` has read-only access to a single data stream, minimising attack surface. Required code changes: 14 lines in the `Executor` module; 75 lines for the new service.

6.4 Action: from issue tracking to collaborative deployment

- Human Action: The lead architect and a compliance officer reviewed the report. The analysis was clear and auditable, and the risk quantified. They gave the 'proceed' command.
- Software Action: Recognitive initiated the changes immediately. It wrote the production-ready code, modified the `Executor` module, updated the system's test harnesses, and provisioned the necessary infrastructure in the staging environment via Infrastructure-as-Code scripts.

The team's role shifted from manual implementation to high-level supervision. They watched the automated pipeline execute, confident in the validated impact

analysis. The process of deploying a secure and compliant solution to staging was reduced from a potential two-week sprint to under four hours.

6.5 Post-action assessment: from hope to certainty

- Staging: The new architecture ran in the staging environment under a 48-hour high-volume stress test.
- Continuous Monitoring: Recognitive observed the results of its own actions, comparing the live telemetry data against the predictions from its earlier simulation.
- Final Report: At the end of the test, it issued a final assessment:

  Post-action analysis complete. Observed latency on `Executor` peaked at 0.07ms, consistent with the <0.1ms simulation prediction. No new 'surprise' events were generated. The generative model has been updated to reflect the new, more accurate system state. The Odyssey platform is now verifiably in compliance with Directive 2095-08B.

6.6 To sum it up

What would normally have been a high-risk and time-sensitive regulatory change is transformed into a demonstration of market-leading agility.

All four parties — the technical lead, the non-technical lead, the AI, and the supersoftware — are 'in the room', allowing Adapt Financial to meet the deadline easily, avoid technical debt, and possess a fully auditable, machine-generated change record.

Recognitive makes the software itself a proactive, reasoning, and self-securing partner in its own evolution.

When a new task arises, there is often a realm of possible solutions. Rapid determination and implementation of the best way forward is a dramatic competitive edge.

The four parties 'in the room' in dialogue.

Prior to LLM-assisted development, had a junior dev proposed Plan A, a senior dev would have rejected it from experience. But this is a slow feedback loop.

With LLMs, Plan A is proposed and it's considered a ‘good enough’ solution absent the expressivity of the software itself. A human would have to intervene, assess the risk, and decide whether to move forward or not.

With Recognitive, the supersoftware expresses why Plan A is a bad idea, and helps the LLM deduce the problem with Plan B. Recognitive pushes the human decision as late as possible, avoiding the ‘Pull Request Fatigue’ that plagues human-LLM collaboration today. It facilitates rapid testing and deployment of Plan C, securing compliance and leaving the system in a robust state with no technical debt.

7. Conclusion

Peter Norvig famously showed that the homoiconic and reflective nature of languages such as Lisp renders many classic design patterns redundant. Such patterns exist to compensate for limitations in less expressive paradigms — they're the carrying costs of the architectural debt.

Modern frameworks have absorbed these patterns, effectively institutionalising the debt to streamline development. While this abstracts away the boilerplate, it locks software into a paradigm of static assembly. We have perfected the mechanics of construction but at the cost of intelligence and adaptability.

The 'Gang of Four' authors of *Design Patterns* were themselves inspired by Christopher Alexander's *Notes on the Synthesis of Form*, yet his later work, *The Nature of Order*, turns decisively from the assembly of parts to the differentiation of wholes. It celebrates the emergence of living systems through processes of unfolding rather than construction.

Think about this in the context of computing.

Recognitive moves software beyond mechanistic assembly towards an ecology of self-referential differentiation.

It's not a library. It's not a standard. It's a grammar of structure and reflection cultivating a new epoch of software characterised by intelligence, speed, resilience, and adaptability. It not only resolves many of today's engineering challenges but holds promise as a foundational component of Artificial General Intelligence (AGI).

8. Our foundations

8.1 Complexity theory

While there is no universally agreed rigorous definition of complexity, it is typically characterised as a system of many interacting components or agents, connected through a dynamic network of relationships. Such systems often exhibit nonlinear interactions, feedback, and history-dependence, giving rise to emergent and sometimes unexpected outcomes. In this context, causal relationships are difficult to isolate — causality is typically distributed, nonlinear, and context-dependent.

All living systems are complex and adaptive. Adaptive means the behaviours of participants in the system change according to the circumstances in which they find themselves.

It's useful to be able to distinguish complexity and complication. A car is complicated for example but not complex, i.e. it consists of very many different integrated parts but does not exhibit unexpected behaviour by design. Traffic on the other hand is not complicated but is complex, i.e. traffic is just a mass of vehicles and traffic jams can emerge for no obvious reason.

Many systems of interest entail both complication and complexity. For such a system to exhibit appropriate adaptability, for it to survive and thrive so to speak, it must exhibit a suitable sophistication to handle the dynamism of its environment. This requisite complexity[36] is perhaps best reflected in the following phrase capturing an idea first expressed by Einstein:

> Everything should be made as simple as possible, but not simpler.[37]

8.2 Bio-analogy and biomimetics

We take nature as our guide. As Aristotle figured out more than two millennia ago:

> If one way be better than another, that you may be sure is nature's way.

You could say nature, including all living processes by definition, has had substantially more time and capacity to try out different things to see what works and what doesn't. Nature is the ultimate R&D lab.

Biomimetics (aka biomimicry) is the purposeful emulation of nature's models, systems, and elements to solve complex human problems. Computer science has long drawn inspiration from natural systems and principles, although mostly via analogical reasoning than biomimicry. For example, while research into biomimetic

neural networks[38] explicitly seeks to replicate biological neuronal processes, today's mainstream neural networks (including LLMs) are better understood as analogies to biological systems rather than faithful emulations.

The line is blurred to the point that biomimetic design can be broadened to span inspiration, imitation, and integration.[39] Either way, the distinctions are instructive. In our contexts here, one may easily identify physical computing infrastructure as analogy (inspiration) and yet observe imitative qualities of the associated algorithmic behaviour, intended or otherwise.

At the risk of confusing matters, researchers observe for example that neural networks can emergently learn to perform analogical reasoning[40], a behavior that emulates a key aspect of human cognition. The design goal may then shift to fostering such emergent outcomes deliberately as best one can given system complexity.

Biological inspiration is abundant in our contexts here.

8.3 Cybernetics

Cybernetics is concerned with regulatory and purposive systems. Core concepts include feedback, variety, viability, and the role of the observer — specifically considering the observer a part of rather than separate from the system in question. It's been described as the science and art of understanding[41], and in terms of having a goal and taking action to achieve that goal.[42]

It's applied in many disciplines including biology, ecology, technology, cognitive sciences, social sciences, and in designing, engineering, learning, managing, music, and conversation. Its application is potentially so broad it's been called antidisciplinary.[43]

Cybernetics and AI have some similarities but:

> AI presumes that value lies in understanding "the world as it is" — which presumes that knowing the world is both possible and necessary. Cybernetics holds that it is only necessary and only possible to be coupled to the world sufficiently to achieve goals, that is, to gain feedback in order to correct actions to achieve a goal.[44]

Cybernetics provides the foundational framework for understanding governance as the steering and control of complex social systems through communication and feedback.

8.4 On governance

Governance is the determination of authority, decision-making, and accountability in the process of organising anything.[45]

Accountability is the obligation to take responsibility for one's actions, decisions, and their outcomes, including the duty to explain and justify them, and face consequences when unable to do so.[46] Accountability fosters ownership, transparency, and responsible behaviour, but we must ensure the agility with which superorganisation can adapt is not impaired in the process.

Any cybernetician will tell you that the purpose of a system is what it does[47], an idea often expressed as POSIWID. It's a provocation well suited to an age in which systems of consequence often exceed the comprehension of any individual or group. Its focus is operational, directing attention to effects rather than intentions, thereby enabling timely intervention whenever outcomes are deemed undesirable or just subpar.

For those familiar with the Cynefin framework[48], social systems play out in the framework's complicated, complex, and chaotic domains. POSIWID reminds us not to trust design intent alone in the complicated domain, but to verify through results. It's integral to operating in the complex domain in which cause and effect may only be clear in hindsight, and being action oriented, POSIWID is wholly supportive of organising in the chaotic domain.

POSIWID doesn't preclude holding system stewards accountable in broader moral or political terms, but it resists deferring action while such processes unfold. Responsibility is shifted from stated aims to observable results, supporting adaptive change even amid ambiguity or dispute. Of course, adaptive change requires control.

8.5 On control

Governance is most effective when it's embedded within the operational dynamics of the system itself rather than imposed externally.

By way of example, Watt's centrifugal regulator, the device spinning above the cylinder of a steam engine that controls the pressure to ensure it doesn't build up dangerously, is an integral and real-time part of the steam engine. It's considerably more effective in terms of responsiveness, stability, and reliability, and so overall system health, than an engine driver keeping an occasional eye on a pressure gauge and making intermittent manual pressure adjustments. It's direct rather than indirect, endogenous rather than exogenous.

In the lexicon of cybernetics, such a direct, endogenous mechanism implies that the system has an awareness of itself. Self-awareness enables a more complex and reflexive form of meaning-making.[49]

8.6 Active inference

Active inference is a framework for understanding how living systems and artificial agents regulate themselves by continuously predicting and minimising the mismatch between expected and actual sensory input — minimising the surprise!

It inherits the idea of self-regulating systems operating through feedback loops and homeostatic control from cybernetics. It formalises these loops as probabilistic inference in a generative AI model, where perception updates beliefs and action changes the world to reduce prediction error.

8.7 Semiotics and umwelt

Semiotics is the study of signs and meaning-making. A sign is anything which 'stands for' (represents) something else and it can take any form.

> Anything has the potential to be interpretable, but nothing has an intrinsic meaning; for human beings, something becomes a sign when it is interpreted as signifying something.[50]

The role of interpreter corresponds to the cybernetic regard for the observer.

The field has two primary traditions, the philosophical (closely related to logic) and the linguistic. Both are applicable in our contexts. It's considered to have three branches: semantics (the meanings of signs), syntactics (the relations between signs), and pragmatics (the use of signs). Again, all are applicable here.

In the philosophical tradition, the understanding or mental concept that a sign creates (known as the interpretant) is itself another sign or signs. In the linguistic tradition, a sign may be said to stand for one thing if only because it doesn't stand for other things. Either way then, semiotics can be thought of as signs in relation to signs.

Computer science conceives of information as something that may be transmitted from A to B. In the context of living systems however, and in accordance with semiotics, we appreciate that an organism isn't the recipient of information transmitted to it but rather a system participant that infers perturbations as being informative; as being a sign.

> The environment contains no information. The environment is as it is.[51]

In other words, information requires an observer. Each observer has an *umwelt*, a subjective experience of the world based on its cognitive capabilities.[52] If any of this paper's authors overhear a conversation in Mandarin for example, our world is unchanged for none of us comprehend the language. We cannot interpret any signs to find meaning. Even if English is spoken, our world will be little affected if we lack the context to understand what is being said.

> Without context, words and actions have no meaning at all.[53]

The construction of meaning is a social imperative. Social systems don't think or feel as one, but they reproduce themselves through communication processes that construct meaning.[54] A meaningful coherence amongst participants is achieved as the words are put into action and the participants appreciate the effects.[55]

We can then conceive of AI systems as new semiotic actors in human cultural ecologies.

8.8 Biosemiotics

Semiotics asks 'How do animals including humans use signs?' In contrast, biosemiotics presents semiosis as a precondition for life not a product of it.

Biosemiotics interprets biological processes, such as cell signaling, animal communication, and genetic expression, as fundamentally sign-based phenomena. Where there is life, there is communication and interpretation.

While it isn't considered mainstream biology in the Kuhnian sense that it isn't incorporated into the dominant paradigm, it engages directly with biological systems (e.g. cells, genes, ecosystems) and aims to understand how living systems function.

The methodological gap between biosemiotics and conventional biology is narrowing. Biologists increasingly recognise semiotic concepts as useful descriptive tools and adopt biosemiotic models as heuristics for understanding biological phenomena such as signaling, regulation, context-sensitivity, and adaptation.

Complexity theory in particular offers a bridge to semiotics, a dynamical substrate, especially in theoretical biology, systems biology, and bioinformatics. It explains how structures and processes arise at all levels — genetic, cellular, organismic — that can function as signs. Self-organising systems emerge that are capable of treating certain patterns not just as causes but as signs — information about something else that guides the system's future state.

8.9 Autopoiesis and organisational closure

What distinguishes the living from the non-living?[56] In each case, a system is living when it's defined by a boundary within which are the systems it needs to regulate and maintain itself — autopoiesis. It includes the idea of organisational closure which means that a system can be viewed as a network that works to maintain itself. In plain language, it forms a whole.

8.10 Cybersemiotics

Cybersemiotics integrates a number of disciplines including cybernetics, biology, semiotics and biosemiotics, autopoiesis, and embodied cognition.

Cybersemiotics draws conceptual analogies from biology and biosemiotics, and we channel it here for heuristic understanding and replication in complex contexts; not as ends in themselves but as means to meanings, as meanings to our end.

8.11 Genetics

An organism's genotype (its genetic constitution) functions primarily as information storage and does not itself contain the machinery to express or process that information. An organism's cellular machinery are expert late binders so to speak, i.e. they 'interpret' genotypic information in the moment, in context, and act on it.

The separation of encoding (DNA) and expertise (proteins) underpins how life systematises encoding and expression through specialised building blocks highly suited for those roles — nucleobases and amino acids respectively. While the DNA provides the instructions, the expression of those instructions can vary due to multiple biological mechanisms in different contexts. The separation affords complexity and evolvability, and with extremely rare exceptions all known cellular life works like this. Ditto supersoftware.

8.12 Artificial (general) intelligence

An influential paper[57] explores two divergent visions of intelligence and so divergent visions of artificial intelligence. One, widely cited in the field, emphasises task-specific performance:

> [AI is] the science of making machines capable of performing tasks that would require intelligence if done by [humans].[58]

The other highlights generality and adaptation:

> AI is the science and engineering of making machines do tasks they have never seen and have not been prepared for beforehand.[59]

This distinction frames current debates. Large language models can generate novel outputs by recombining patterns from training, but they do not exhibit robust generalisation to out-of-distribution tasks. With LLM performance effectively defining AI today, the more demanding criteria are now associated with the prospect of artificial general intelligence (AGI), for example:

> [AGI] demands a fundamental shift towards systems capable of genuine fluid intelligence, the ability to adapt to novel challenges and solve problems efficiently, much like humans do.[60]

9. Appendix — example domain-agnostic algorithm

```
type Filter<E extends Entity> = (e: E) => boolean

/** recursively collects all entities of a given type from a data graph */
function collect<E extends Entity>(type: EntityType<E>, data: Base,
    filter: Filter<E> = () => true,
    res = new Set<E>, visited = new Set<E>): Set<E> {
    if (data == null)
            return res

    // handle types using reflection
    switch (data.TYPE.kind()) {
            case TypeKind.ENTITY:
                const entity = data as Entity

                // skip recurrence
                if (visited.has(entity))
                        break

                // traverse all property values using reflection
                        visited.add(entity)

                // polymorphic check for matches
                if (type.isAssignableFrom(entity.TYPE) && filter(entity))
                        res.add(entity)

                // traverse all property values using reflection
                for (const property of entity.TYPE.properties())
                        collect(type, property.get(entity), res, visited)

                break
        case TypeKind.LIST:
        case TypeKind.SET:
                // Lists and Sets are handled as Iterable
```

```
                const iterable = data as Iterable<Base>

                for (const element of iterable)
                        collect(type, element, res, visited)

                break
        case TypeKind.MAP:
                const map = data as Map<Base, Base>

                for (const key of map.keys())
                        collect(type, key, res, visited)

                for (const value of map.values())
                        collect(type, value, res, visited)

                break
        }

    return  res;
}

// example of collect() usage
const companies = queryCcompanies()
const persons = collect(Person, companies, e => e.name.startsWith('Ada'))
const addresses = collect(Address, persons)
```

10. References

[1] See https://research.ibm.com/topics/neuro-symbolic-ai

[2] See https://github.com/google-deepmind/alphageometry

[3] See https://learn.microsoft.com/en-us/semantic-kernel/overview/

[4] Please note that the repository documentation isn't aligned with this white paper at the time of writing. See https://github.Com/hiconic-os

[5] Brooks 1980, Technology, Evolution, And Purpose, https://www.jstor.org/stable/20024649

[6] Maturana and Varela 1972, Autopoiesis and Cognition: The Realization of the Living

[7] Newen et al 2018, The Oxford handbook of 4E cognition, https://doi.org/10.1093/oxfordhb/9780198735410.001.0001

[8] Shapiro & Spaulding 2025, Embodied Cognition, The Stanford Encyclopedia Of Philosophy, https://plato.stanford.edu/entries/embodied-cognition/

[9] Marcus 2025, Generative AI's Crippling And Widespread Failure To Induce Robust Models Of The World, https://garymarcus.substack.com/p/generative-ais-crippling-and-widespread

[10] Vafa et al 2025, What Has A Foundation Model Found? Using Inductive Bias To Probe For World Models, https://arxiv.prg/pdf/2507.06952

[11] Jylin04 et al 2024, OthelloGPT Learned A Bag Of Heuristics, https://www.lesswrong.com/posts/gcpnueznxapayakby/othellogpt-learned-a-bag-of-heuristics-1

[12] Gu et al 2025, Challenges And Paths Towards Ai For Software Engineering, https://arxiv.org/pdf/2503.22625

[13] A Type 2 integration per the NeSy taxonomy presented in Wang et al 2024, Towards Data-and Knowledge-Driven Artificial Intelligence: A Survey on Neuro-Symbolic Computing, http://arxiv.org/abs/2210.15889

[14] Jansen And Bosch 2005, Software Architecture As A Set Of Architectural Design Decisions, https://doi.org/10.1109/wicsa.2005.61

[15] Hofstadter 1979, Gödel, Escher, Bach: An Eternal Golden Braid, https://en.wikipedia.org/wiki/G%C3%B6del,_Escher,_Bach

[16] Farnsworth et al 2013, Living Is Information Processing: From Molecules To Global Systems, https://arxiv.org/pdf/1210.5908

[17] Quantum Science Techscribe 2025, Lisp And The Dawn Of Artificial Intelligence: A Historical And Contemporary Perspective, https://quantumzeitgeist.com/lisp-and-the-dawn-of-artificial-intelligence/

[18] De La Torre 2025, From Tool Calling To Symbolic Thinking: Llms In A Persistent Lisp Metaprogramming Loop, https://arxiv.org/html/2506.10021v1

[19] Quantum Science Techscribe 2025, Lisp And The Dawn Of Artificial Intelligence: A Historical And Contemporary Perspective, https://quantumzeitgeist.com/lisp-and-the-dawn-of-artificial-intelligence/

[20] Lampinen & McClelland 2020, Transforming Task Representations To Perform Novel Tasks, https://doi.org/10.1073/pnas.2008852117

[21] See: Von Foerster 1970, Thoughts And Notes On Cognition, https://link.springer.com/chapter/10.1007/0-387-21722-3_5, and Pask 1972, A Fresh Look At Cognition And The Individual, https://www.sciencedirect.com/science/article/abs/pii/S0020737372800026

[22] Chollet 2019, On The Measure Of Intelligence, https://arxiv.Org/abs/1911.01547

[23] Harnad 1990, The Symbol Grounding Problem, https://doi.org/10.1016/0167-2789(90)90087-6

[24] Roy et al 2021, From Machine Learning to Robotics: Challenges and Opportunities for Embodied Intelligence, https://arxiv.org/pdf/2110.15245

[25] Ndea homepage, https://ndea.com, retrieved September 16th, 2025

[26] Kimovski et al 2023, Beyond Von Neumann in the Computing Continuum: Architectures, Applications, and Future Directions, https://ieeexplore.ieee.org/abstract/document/10207712/

[27] Becker et al 2025, Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity, https://arxiv.org/abs/2507.09089

[28] Recht 2025, Are developers finally out of a job?, https://www.argmin.net/p/are-developers-finally-out-of-a-job

[29] See https://modelcontextprotocol.io

[30] Quantum Science Techscribe 2025, LISP and the Dawn of Artificial Intelligence: A Historical and Contemporary Perspective, https://quantumzeitgeist.com/lisp-and-the-dawn-of-artificial-intelligence/

[31] Njanka et al 2021, IT-Business Alignment: A Systematic Literature Review, https://www.sciencedirect.com/science/article/pii/S1877050921001940

[32] Ibid. 31

[33] Winner 1980, Do artefacts Have Politics?, https://doi.org/10.1177/030631299029003004

[34] Technology Is Not Values Neutral: Ending the Reign of Nihilistic Design, The Consilience Project 2022, https://consilienceproject.org/technology-is-not-values-neutral-ending-the-reign-of-nihilistic-design-2/

[35] See https://vsdesign.org

[36] A concept attributed to Fredmund Malik (Management: Mastering Complexity, 2012), extending Ashby's Law of Requisite Variety

[37] See https://quoteinvestigator.com/2011/05/13/einstein-simple/

[38] Neuromorphic computing for example.

[39] Gerola et al 2023, https://link.springer.com/article/10.1007/S13347-023-00665-0

[40] Lampinen, Shaw, & McClelland 2017, Analogies Emerge from Learning Dyamics [Sic] in Neural Networks, https://escholarship.org/uc/item/5s8259wx

[41] Maturana 1999, https://asc-cybernetics.org/wavefront/contributes/perspectives.htm

[42] Pangaro, https://pangaro.com/definition-cybernetics.html

[43] Pickering 2008, Emergence and synthesis: Science studies, cybernetics and antidisciplinarity, https://intellectdiscover.com/content/journals/10.1386/tear.6.2.127_1

[44] Ibid. 42

[45] Chartered Governance Institute Uk & Ireland, What is Governance?, retrieved 1 August 2025, https://www.cgi.org.uk/resources/factsheets/factsheets/what-is-governance/

[46] Bovens 2007, Analysing and Assessing Accountability: A Conceptual Framework, https://doi.org/10.1111/j.1468-0386.2007.00378.x

[47] Beer 1985, Diagnosing the system for organizations

[48] See https://thecynefin.co/about-us/about-cynefin-framework/

[49] Gallagher 2023, Sense-Making (6.4), Embodied and Enactive Approaches to Cognition, https://doi.org/10.1017/9781009209793

[50] Chandler 2022, Semiotics: The Basics, 4th edition, https://www.taylorfrancis.com/books/mono/10.4324/9781003155744/semiotics-basics-daniel-chandler

[51] Von Foerster 1970, Thoughts and notes on cognition, https://link.springer.com/chapter/10.1007/0-387-21722-3_5,

[52] Von Uexküll 1934, Streifzuge durch die umwelten von Tieren und Menschen Ein Bilderbuch unsichtbarer Welten: Einundzwanzigster Band

[53] Bateson 1979, Mind and Nature: A Necessary Unity

[54] Luhmann 1995, Social Systems, https://www.sup.org/books/sociology/social-systems

[55] Ibid. 51

[56] Ibid. 6

[57] Ibid. 22

[58] Minsky 1968, as quoted in Hernández-Orallo 2017, Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement, https://link.springer.com/article/10.1007/s10462-016-9505-7

[59] Hernández-Orallo 2017 paraphrasing McCarthy, Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement, https://link.springer.com/article/10.1007/s10462-016-9505-7

[60] Ibid. 22

###