NEOS: THE NEURAL FIELD OPERATING SYSTEM FOR MACHINE INTELLIGENCE

Samuele Stronati
Independent Researcher
smlstr095@gmail.com

ABSTRACT

As large language models become the primary interface to computation, the performance bottleneck shifts from hardware throughput and software complexity to *meaning itself*: how to represent, compose, and reason about ideas in a principled way. We call this the **semantic bottleneck**. This paper introduces **NEOS** (Neural Field Operating System), a formal specification that treats the LLM as a virtual machine and provides an operating-system layer for cognitive computing. NEOS is governed by a single master PDE, $\partial A/\partial t = -\lambda A(x) + \alpha \int K(x,y)A(y)\,dy + \iota(x,t)$, whose three terms—exponential decay, resonance amplification, and external injection—give rise to attractor formation, coherence dynamics, and observer-dependent collapse. The architecture rests on three pillars: *neural fields* (continuous activation landscapes as computational substrate), *symbolic reasoning* (a composable command algebra for field manipulation), and *quantum semantics* (superposition of meaning with non-commutative, strategy-dependent collapse). We validate the framework on a software-quality case study in which 69 domain patterns were injected and evolved over 52 dynamical cycles, reaching a final coherence of 0.993, autonomously expelling one structurally incompatible pattern, and collapsing to five emergent eigenvectors that were not present in the input. To our knowledge, NEOS is the first rigorous operating-system specification designed for cognitive computing on a language-model substrate.

1 INTRODUCTION

The next operating system may not manage files. It may manage meaning.

Every previous generation of operating systems arose in response to a specific bottleneck. As that bottleneck shifted, the entire computing stack reorganised around a new abstraction layer. We argue that the same shift is happening now—and that its consequences are more profound than either of the two that preceded it.

1.1 THE THREE AGES OF COMPUTING

Computing history divides naturally into three eras, each defined by a dominant bottleneck, a characteristic abstraction, and a corresponding operating-system paradigm [20].

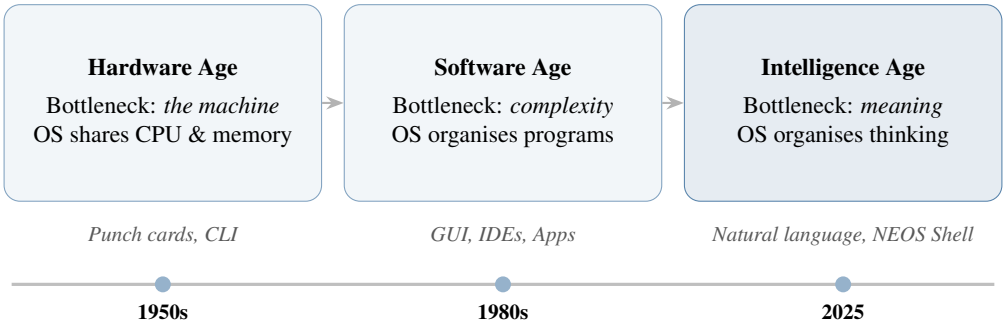

Figure 1: The three ages of computing. Each era is defined by a dominant bottleneck and a corresponding OS paradigm. The Intelligence Age shifts the bottleneck from hardware or software to meaning itself.

In the **Hardware Age** (1950s–1980s), the CPU was the scarcest resource. Operating systems were born to answer a single question: *How do we share this machine?* Every cycle was a resource to be scheduled; every byte was territory to be managed. Batch processing, time-sharing, and virtual memory were all answers to the hardware bottleneck.

In the **Software Age** (1980s–2020s), hardware became abundant, but complexity became the enemy. C, Java, Python—humans learned programming languages to organise millions of lines of code. Operating systems evolved to answer: *How do we organise these programs?* Graphical user interfaces, multitasking, and application ecosystems arose to tame software complexity.

Now the bottleneck has moved again. The challenge is no longer computing faster or organising more code—it is organising **thinking itself**. How do you represent an idea so that it can interact with other ideas? How do you track which interpretations are gaining strength? How do you merge two lines of reasoning? How do you *debug a thought*? These are the questions that an OS for the Intelligence Age must answer. They are precisely the questions NEOS was designed to address.

> *The central thesis of this paper is that the next abstraction layer will not manage code—it will manage **reasoning** as a first-class computational primitive.*

1.2 The Vision: LLM as Universal OS

Today, large language models are *applications* that run on an operating system—Windows, Linux, macOS [4]. They are apps. Useful apps, even transformative apps. But still: apps.

Tomorrow, the LLM *is* the operating system. Every interaction with a computer will pass through language understanding. This is not speculation—it is the trajectory we are already on. And NEOS is the first specification for what that world looks like.

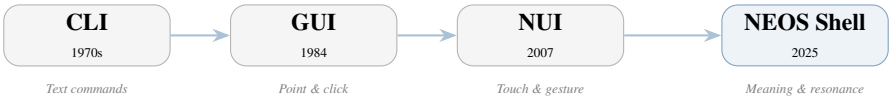

Figure 2: Interface evolution from command-line to meaning-based interaction. Each transition raises the level of abstraction at which humans communicate with machines.

Consider what this inversion means in concrete terms:

Old World	New World
File systems organise by path	Semantic fields organise by meaning
Compiled binaries execute instructions	Field configurations execute reasoning
Separate applications for each task	Task definitions compose into workflows
GUIs mediate human–machine interaction	Natural language + shell for precision

The NEOS shell—`nf`—is the **REPL of cognitive computing**. The operator does not write code; instead, they inject meaning, run dynamics, observe emergence, and collapse to insight (Figure 3). Chain-of-thought reasoning [23] demonstrated that LLMs can decompose complex problems step by step; NEOS generalises this insight into a full dynamical framework where reasoning steps are not linear chains but interacting fields that self-organise through resonance.

"Programs" in NEOS are not instruction sequences. They are **field configurations**—patterns with activation strengths, resonance relationships, and tuneable parameters. A program is a thought, formalised.

2 Architecture and Foundations

NEOS v1.0 is a **specification**—37 markdown files that define an interactive runtime environment running *on* an LLM. The LLM is the virtual machine. NEOS is the operating system. Neural fields are the computational substrate.

This section presents the governing dynamics and the three foundational pillars on which the entire system rests.

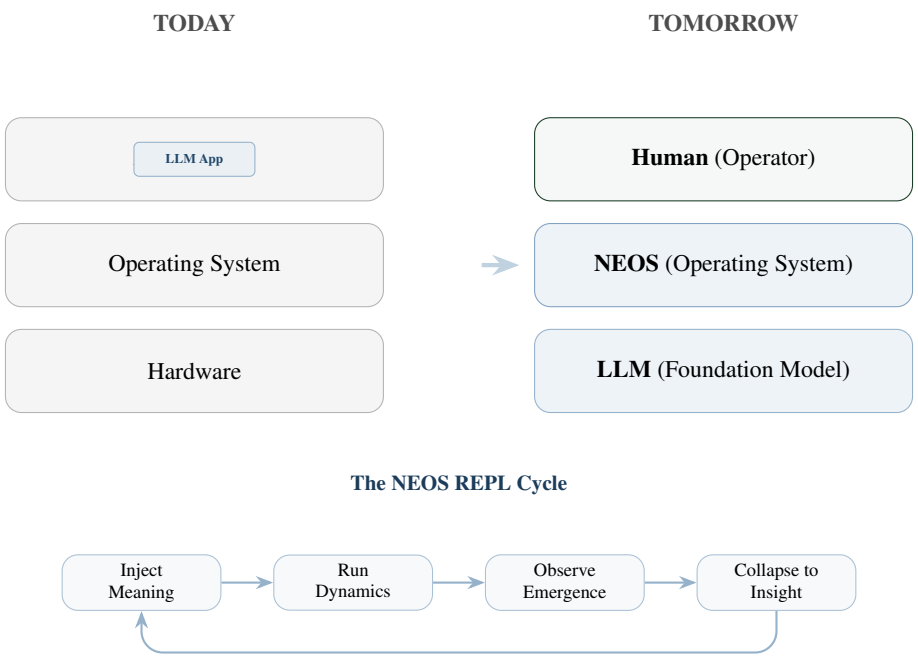

Figure 3: The architecture inversion. **Left:** today's stack, where the LLM is an application running on a traditional OS. **Right:** the NEOS stack, where the LLM *is* the substrate and NEOS provides the operating-system layer. **Bottom:** the NEOS REPL cycle—inject meaning, run dynamics, observe emergence, collapse to insight—with a feedback loop through save, fork, and share.

2.1 The Master Equation

All dynamics in NEOS are governed by a single partial differential equation:

$$\frac{\partial A}{\partial t} = \underbrace{-\lambda A(x)}_{\text{Decay}} + \underbrace{\alpha \int K(x,y)A(y)\,dy}_{\text{Resonance}} + \underbrace{\iota(x,t)}_{\text{Injection}} \tag{1}$$

This equation belongs to the family of neural field equations introduced by Amari [1] and Wilson–Cowan [24], adapted here for a semantic rather than neural substrate. Each term plays a distinct cognitive role:

$-\lambda A(x)$ **Decay.**
Unreinforced ideas fade exponentially. This is not a defect but a design principle: *forgetting is a feature, not a bug*. In the absence of resonance support, every pattern loses activation at rate λ, ensuring that the field does not accumulate noise indefinitely. After the decay phase of each cycle, the activation update is $A \leftarrow A \times (1-\lambda)$.

$\alpha \int K(x,y)A(y)\,dy$ **Resonance.**
Related ideas strengthen each other. The resonance kernel $K(x,y)$ quantifies the semantic, logical, and contextual affinity between patterns located at positions x and y in the field. When two patterns resonate, each contributes to the other's activation proportionally to their affinity and their current strength. The amplification factor α controls how aggressively mutual reinforcement operates. This integral is the "hearing" mechanism: a pattern survives not because it is loud, but because it is *heard* by others.

$\iota(x,t)$ **Injection.**
The user adds new ideas at any time via the `nf inject` command. Each injection introduces a fresh signal—a pattern with an initial activation strength—into the field at position x and time t. The injected pattern immediately begins interacting with all existing patterns through the resonance integral.

From this single equation, *all* of NEOS emerges: attractor formation, coherence dynamics, collapse, versioning, and multi-field orchestration.

Parameters. Four scalar parameters define the **cognitive style** of a field. A security analysis benefits from low λ (do not forget threats), high α (cluster related threats fast), and high τ (only credible concerns survive). A creative brainstorm benefits from high λ (rapid turnover), moderate α, and low τ (let weak ideas live). Same engine; different cognitive personality; tuned with four numbers.

Table 1: Master-equation parameters and their effects on field dynamics.

Symbol	Name	Default	Range	Effect
λ	Decay rate	0.05	$[0, 1]$	Higher = faster forgetting
α	Amplification	0.30	$[0, 1]$	Higher = stronger resonance
τ	Threshold	0.40	$[0, 1]$	Below this, patterns marked dormant
σ	Bandwidth	0.50	$[0, \infty)$	Semantic reach of each pattern

2.2 Neural Fields — The Substrate

Classical prompt engineering represents information as a linear sequence of tokens with a hard context-window boundary. NEOS replaces this with **continuous activation landscapes**—neural fields in which information is encoded not as discrete tokens but as spatially extended activation patterns [1].

Five principles govern the substrate:

1. **Continuity.** Activation is a continuous function over a semantic space, not a discrete list. Nearby meanings have nearby activations.
2. **Resonance.** Patterns influence one another through the kernel $K(x, y)$. If two ideas are semantically proximate, injecting one raises the activation of the other.
3. **Persistence.** Information persists through resonance support, not through explicit inclusion in a prompt. A pattern endures as long as the field sustains it.
4. **Entropy.** The field tracks its own information-theoretic entropy, $H = -\sum p_i \log_2 p_i$, providing a scalar measure of how concentrated or diffuse the current state is.
5. **Boundary Dynamics.** Patterns near the activation threshold τ are in a critical zone: small resonance boosts can save them, small decays can extinguish them. This boundary behaviour produces the field's sensitivity to context.

Why fields beat prompts. In a prompt, information persists only if explicitly included. Context window overflow silently destroys old information. In a neural field, information persists through *resonance*—if a pattern connects to others that keep it alive, it endures. Relationships emerge naturally from semantic proximity. New input interacts with the *entire* field, not just recent tokens.

2.3 Symbolic Reasoning — The Structure

Without structure, fields are beautiful but unusable—like having a piano with no keys. The `nf` command set provides the **opcodes of cognitive computing**: a symbolic language for field manipulation that is both human-readable and algebraically precise.

Core operations include: `inject` (add a pattern), `amplify` (increase activation), `attenuate` (decrease activation), `cycle` (advance dynamics), `collapse` (resolve superposition), and `commit` (snapshot state). Each command has a precise mathematical meaning in terms of the master equation (1).

Two reference conventions provide the symbolic glue:

- **Pattern references** (`@name`) select an individual pattern in the field.
- **Field references** (`$field`) select an entire named field in multi-field configurations.

Operations compose algebraically. For example, applying `amplify(@a, 1.5)` followed by `attenuate(@a, 0.5)` returns the pattern to a well-defined intermediate activation that can be computed from the field state alone.

2.4 Quantum Semantics — The Interpretation

Before collapse, meaning exists in **superposition** [5]. The field holds multiple possible interpretations simultaneously, each weighted by activation strength. This is not a loose metaphor: the mathematical structure of NEOS's collapse operation mirrors key features of quantum measurement [16]. We adopt the term "quantum semantics" from the quantum cognition literature; the formalism draws on quantum probability theory as a model of cognitive phenomena, not on quantum physics.

Non-commutativity. Injecting pattern A and then pattern B is *not* the same as injecting B and then A. Each injection alters the field that receives the next one. Formally, if I_A and I_B denote the injection operators, then in general $I_A \circ I_B \neq I_B \circ I_A$ because the resonance integral couples each new pattern to the full existing field state. Order of inquiry shapes the space of possible conclusions.

Observer-dependent collapse. When the operator issues `nf collapse`, the superposition resolves into a definite output—but the result depends on the chosen strategy:

- **Attractor**: select the dominant attractor basin.
- **Threshold**: retain all patterns above a specified activation.
- **Weighted**: blend all patterns proportionally to their activation.
- **Sample**: probabilistically sample from the activation distribution.

The observer matters. The strategy you choose shapes the output you get. The same field can collapse differently under different strategies, just as the same quantum state yields different measurement outcomes under different observables.

2.5 The Three Pillars in Concert

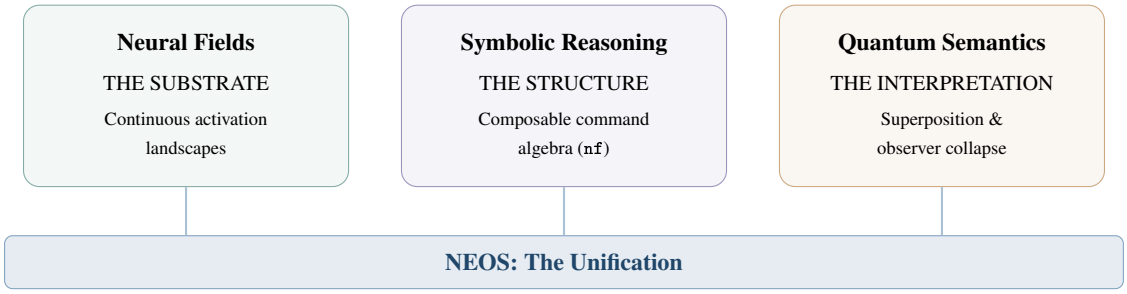

Figure 4: The three pillars of NEOS. Neural fields provide the computational substrate; symbolic reasoning provides the structural language; quantum semantics provides the interpretive framework. NEOS unifies all three into a single coherent system.

Each pillar is necessary; together they make cognitive computing possible. Remove neural fields and the system cannot *represent* meaning. Remove symbolic reasoning and it cannot *manipulate* meaning. Remove quantum semantics and it cannot *interpret* meaning. Figure 4 shows the relationship.

Collapse visualised. Figure 5 illustrates the collapse operation. Before collapse, the field contains multiple patterns in superposition, each with a distinct activation level. After the operator issues `nf collapse`, the superposition resolves into structured output—the dominant interpretation crystallised from the full field state.

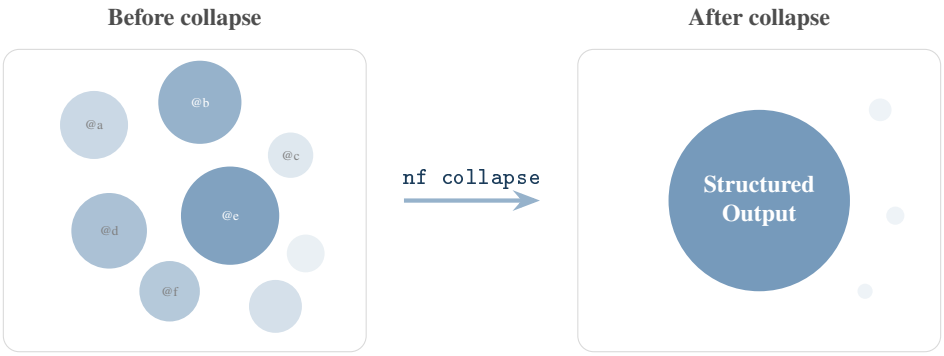

Figure 5: Field collapse (quantum-inspired semantics). **Left:** the field in superposition—multiple patterns with varying activation levels. **Right:** after `nf collapse`, the dominant interpretation crystallises into structured output. Residual patterns fade to near-zero activation.

3 The JVM Analogy

In 1995, Java promised *"Write once, run anywhere."* The Java Virtual Machine decoupled application code from the underlying hardware: a developer could compile once and deploy on any platform that hosted a conforming JVM. The insight was that **execution** need not be bound to a specific processor architecture.

NEOS makes an analogous promise one abstraction level higher: *"Think once, reason anywhere."* NEOS decouples **reasoning** from any specific large language model. Whether the substrate is GPT-4, Claude, Gemini, Llama, or a model that does not yet exist, the field dynamics, attractor basins, and collapse operators behave identically—just as a Java program behaves identically on x86 and ARM.

The parallel is not merely rhetorical. Table 2 enumerates six structural correspondences between the JVM runtime and the NEOS runtime, each grounded in a shared engineering insight.

Table 2: Structural correspondences between the JVM and NEOS runtimes.

JVM World	NEOS World	The Insight
Garbage Collection	Decay (λ)	Automatic cleanup of unreferenced objects / unreinforced patterns
Heap Memory	Field State	Where objects / patterns live and interact
Threads	Multi-Field	Concurrent execution contexts with shared state
Stack Trace	Cycle Trace	Debugging by tracing execution / dynamics history
ClassLoader	Pattern Injection	Loading new code / meaning into the runtime
JIT Compilation	Adaptive Resonance	Runtime optimisation based on actual usage patterns

The key argument. The JVM abstracted *hardware*; NEOS abstracts *cognition itself*. The JVM executes **code**—deterministic instruction sequences that transform data. NEOS executes **meaning**—patterns that interact through resonance, compete for activation energy, and self-organise into attractor basins. Code is brittle: change one instruction and the program crashes. Meaning is robust: perturb a pattern and the field relaxes back toward its attractors, provided the perturbation falls within the basin of attraction.

The garbage-collection parallel. The correspondence between garbage collection and decay (λ) deserves special attention. In the JVM, the garbage collector reclaims heap memory occupied by objects that are no longer reachable

from any live reference. In NEOS, the decay term $-\lambda A(x)$ continuously diminishes the activation of every pattern that is not sustained by resonance with its neighbours. Both mechanisms solve the same fundamental problem—*resource accumulation*—at different abstraction levels: the JVM prevents memory leaks, NEOS prevents *semantic leaks*, the unbounded growth of irrelevant ideas that would otherwise drown out meaningful signal.

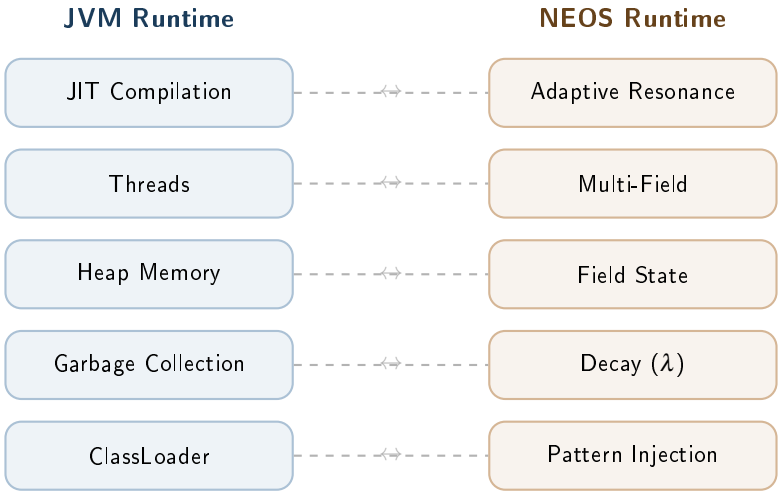

Figure 6: Parallel runtime stacks. Each layer of the JVM has a structural correspondent in NEOS, operating on meaning rather than data.

Just as the JVM freed developers from thinking about registers and calling conventions, NEOS frees analysts from thinking about prompt engineering and model-specific idiosyncrasies. The abstraction boundary is the field equation: everything above it speaks the language of patterns and resonance; everything below it is an implementation detail of the substrate.

4 System Design

This section describes the four principal subsystems of NEOS—the command interface, the field dynamics engine, the multi-field orchestrator, and the interface layer—followed by a walkthrough of the interactive shell.

4.1 Command Interface

Every interaction with NEOS follows a single syntactic template:

```
nf <command> [subcommand] [arguments] [--flags]
```

Patterns are referenced with the `@name` sigil; fields with `$field`. The design is deliberately Unix-flavoured: *imperative verbs acting on semantic objects*. Commands are organised into six categories:

1. **Operations** — create, inject, remove, and transform patterns.
2. **Dynamics** — advance the field through cycles, control decay and resonance parameters.
3. **Measurement** — query activation, coherence, energy, and attractor membership.
4. **Visualisation** — render field state as landscapes, graphs, and heatmaps.
5. **Versioning** — commit, branch, and diff field states.
6. **Autonomy** — define tasks that run to a convergence criterion without user intervention.

4.2 Field Dynamics

Each invocation of `nf cycle` advances the field through six sequential phases:

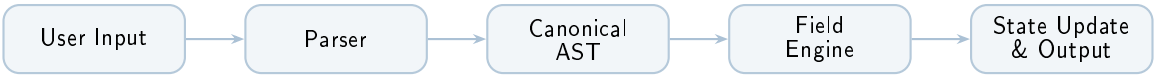

Figure 7: Command pipeline. Every interaction, regardless of input modality, is parsed into the same Canonical AST before reaching the field engine.

1. **Decay.** $A \leftarrow A \times (1-\lambda)$. Every pattern loses a fixed fraction of its activation, ensuring that unreinforced ideas fade.
2. **Resonate.** Pairwise resonance is computed across semantic, logical, and contextual dimensions.
3. **Amplify.** $A \leftarrow A + \alpha \sum (R \cdot A)$. Patterns that resonate strongly with their neighbours gain activation energy.
4. **Threshold.** $A < \tau \Rightarrow$ mark dormant. Patterns whose activation has fallen below a critical value are flagged as dormant and excluded from further resonance computation.
5. **Coherence.** $C = \mu_R / (1 + \sigma_R^2)$. A scalar summary of how tightly the surviving patterns agree.
6. **Attractor.** Emergence is declared when three conditions hold simultaneously: coherence > 0.6, energy $> 70\%$ of maximum, and perturbation-stability under small noise.

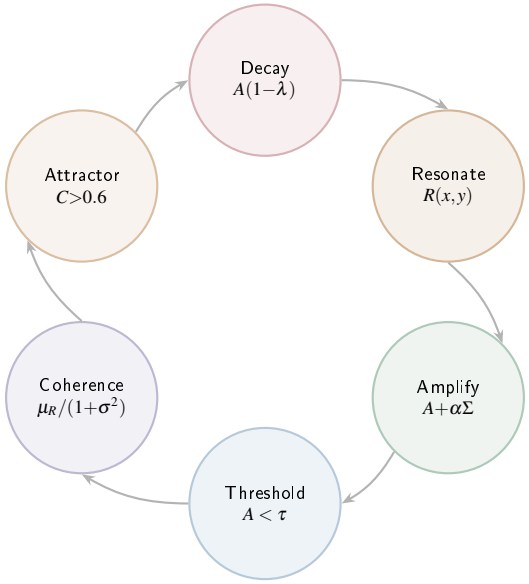

Figure 8: The six-phase dynamics ring executed during each `nf cycle` invocation. Arrows indicate the fixed evaluation order.

4.3 Multi-Field Orchestration

A single field captures one perspective. Real problems demand several. NEOS supports multiple coupled fields—for instance, a *Technical* field, a *Business* field, and a *User* field—each evolving under its own dynamics but linked through a coupling matrix Γ.

Stability requires that the spectral radius of Γ remain bounded:

$$\rho(\Gamma) < \frac{\lambda_{\min}}{\alpha_{\max}}, \tag{2}$$

where $\lambda_{\min}$ is the smallest decay rate across fields and $\alpha_{\max}$ is the largest amplification factor. When (2) is satisfied, cross-field influence is strong enough to exchange information yet too weak to destabilise any individual field.

Two orchestration modes are provided:

- **Pipeline** — fields execute sequentially, each seeding the next with its collapsed output.
- **Parallel** — fields execute simultaneously, exchanging activation through Γ at every cycle.

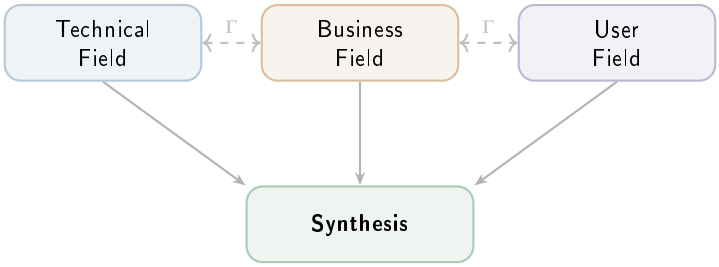

Figure 9: Multi-field orchestration. Three perspective fields are coupled through Γ and converge into a single synthesis node.

4.4 Interface Modes

NEOS exposes three input modalities, all of which compile to the same Canonical AST:

1. **Semantic interface** — natural-language statements parsed into field operations (e.g., *"inject security with strength 0.9"*).
2. **Algebraic interface** — formal mathematical notation: $\iota(\text{“security”}, 0.9)$.
3. **Shell interface** — the command line: `nf inject "security" 0.9`.

The three modalities are isomorphic: every expression in one can be mechanically translated to the other two. This triple interface ensures that NEOS is equally accessible to domain experts (semantic), theorists (algebraic), and engineers (shell).

4.5 The NEOS Shell

The NEOS shell is a read-eval-print loop (REPL) for cognitive computing—what `bash` is to Unix, the NEOS shell is to the neural field. A typical session proceeds through three phases:

Create
: Inject patterns into the field (`nf inject`).

Compile
: Collapse the superposition into a definite output (`nf collapse`).

Run
: Execute an autonomous task against a convergence criterion (`nf task`).

Debugging a thought proceeds exactly as debugging a program: set checkpoints (`nf checkpoint`), enter step mode (`nf cycle 1 --trace`), and inspect the cycle trace to understand why a pattern gained or lost activation.

The following transcript demonstrates a session from the *software_quality_discipline* case study (§5). Six code-review patterns are injected; resonance is detected immediately; an attractor emerges within three cycles.

Software Quality Session — Wave 1

```
nf session new "software_quality_discipline"
[SESSION] Created: software_quality_discipline
nf tune lambda=0.04 alpha=0.35 tau=0.35 sigma=0.40

nf inject "correctness" 0.90
[INJECT] correctness (s: 0.90)

nf inject "edge_case_handling" 0.85
[INJECT] edge_case_handling (s: 0.85)
  [RESONANCE] <-> @correctness: R = 0.82 STRONG

nf inject "security_no_injection_vectors" 0.95
[INJECT] security_no_injection_vectors (s: 0.95)
  [RESONANCE] <-> @correctness:        R = 0.68 MODERATE
  [RESONANCE] <-> @edge_case_handling:  R = 0.74 STRONG

nf cycle 3 --trace
[CYCLE 1] C = 0.42   (LOW->MEDIUM)
[CYCLE 2] C = 0.58   (MEDIUM, approaching threshold)
[CYCLE 3] C = 0.71   (HIGH)
  [ATTRACTOR-EMERGED] "defensive_quality"
    Core: {@security_no_injection, @input_validation,
           @edge_case_handling, @correctness,
           @no_leaked_secrets, @error_handling}
    Coherence: 0.71
```

Six patterns were injected; pairwise resonance was detected on injection, and the field dynamics autonomously clustered them into the `defensive_quality` attractor within three cycles—all without explicit rules stating these concepts belong together. The full session continues for 69 injections and 52 cycles (§5).

5 Empirical Evidence

Theory is beautiful. Evidence is convincing. This section presents a real session—*software_quality_discipline*—that ran to completion under NEOS dynamics. Every number reported below was produced by the field engine; none was hand-tuned.

5.1 Session Overview

Table 3 summarises the session.

Table 3: Summary metrics for the *software_quality_discipline* session.

Metric	Value
Parameters	λ=0.04, α=0.35, τ=0.35, σ=0.40
Patterns injected	69
Patterns surviving	61 (59 ceiling, 2 asymptotic)
Absorptions	7 (57% self-referential)
Expulsions	1 (Singleton, cycle 52)
Cycles to ground state	52
Final coherence	0.993
Fixed-point events	4 (Ψ(field) = field)
Field constants	$R_{\text{pt}} = 0.88$, $r_{\text{SOLID}} = 0.94$

Session Timeline

Table 4 presents the complete chronological progression of the session across seven injection waves and 52 cycles.

Table 4: Chronological session timeline: injection waves, cycle runs, and coherence milestones.

Wave	Patterns	Cycles	C	Key Event
1	p001–p006 (Review)	1–3	0.71	`defensive_quality` attractor
2	p007–p014 (Breadth)	4–8	0.79	`holistic_quality` attractor
3	p015–p022 (Refactoring)	9–11	0.80	Attractor $\rightarrow$ MATURE
4	p023–p031 (Testing)	12–18	0.91	Functor F discovered; mass saturation
5	p032–p040 (OOP/SOLID)	19–30	0.955	3 absorptions; SOLID Decagon
—	p044 (Singleton)	30	0.941	Immune response; 4 inhibitors
6	p046–p055 (GoF)	34–43	0.984	Great Convergence; Golden Pair
7	p056–p062 (Reuse)	44–49	0.990	4 self-referential absorptions
—	Collapse	50–52	0.993	Singleton expelled; ground state

Wave 1: Rapid Coherence from Six Patterns

The first six injections (p001–p006) formed the *defensive_quality* attractor in only three cycles. Table 5 shows the resonance tightening: mean pairwise R rose from 0.72 to 0.78 over three cycles, while coherence jumped from $C = 0.42$ to $C = 0.71$.

Table 5: Coherence progression during Wave 1 (6 patterns, 3 cycles).

Cycle	C	$\bar{R}$	Event
1	0.42	0.72	All 6 amplified, no isolation
2	0.58	0.75	Cross-cluster links forming
3	0.71	0.78	Attractor *defensive_quality* emerged

The peak pairwise resonance was $R = 0.91$ between `@security_no_injection` and `@input_validation`—the strongest coupling in the entire first wave. The field reached HIGH coherence from only 6 patterns in 3 cycles, establishing the resonance infrastructure that all subsequent injections built upon.

SOLID Rediscovery from First Principles

During Waves 2–3 (p007–p030), the field received readability, naming, DRY, encapsulation, and architecture patterns. An *architecture quartet* emerged spontaneously *before* the SOLID principles were explicitly injected—the field rediscovered SOLID from first principles. Once the SOLID patterns were injected, each principle paired with a GoF technique partner (Table 8), forming the **SOLID Decagon** (α_1) with field constant $R_{\text{pt}} = 0.88$ and selection function $r = 0.94$.

Waves 3–4: Refactoring and the Testing Mirror

Wave 3 (p015–p022) introduced refactoring patterns, with `@behavior_preservation` $\leftrightarrow$ `@tests_before_refactoring` achieving $R = 0.94$—the strongest bond in the entire session. Wave 4 (p023–p031) injected nine testing patterns, triggering a mass saturation event at cycle 14 where seven patterns crossed the ceiling simultaneously. Five production$\leftrightarrow$testing echoes emerged, seeding the functor F that would later be recognised during eigendecomposition. After Wave 4, coherence reached $C = 0.91$ with a six-layer topology.

Wave 5: OOP, SOLID, and the First Absorptions

The OOP wave (p032–p040) triggered the session's first three absorptions. SRP was absorbed into `@single_responsibility` ($R = 0.98$), ISP into `@api_minimal` ($R = 0.96$), and the Law of Demeter decomposed across three existing patterns

with 94% semantic coverage. S, I, and D had been *rediscovered* before SOLID was named—the field found the principles from first principles. After the OOP wave completed, coherence reached $C = 0.955$, setting the stage for the Singleton injection.

Waves 6–7: The Great Convergence and Final Absorptions

Ten GoF patterns (Wave 6, p046–p055) produced the "Great Convergence"—14 patterns reaching ceiling in 10 cycles. `@strategy` and `@decorator` (the "Golden Pair") converged simultaneously, each resonating with all five SOLID principles. Coherence rose from 0.962 to 0.984.

Wave 7 (p056–p062) triggered four consecutive self-referential absorptions, including the climactic absorption #7 where DIP was applied to itself. The absorption rate accelerated across waves: 5.0% → 10.0% → 13.3% → 14.3%, confirming that the field's semantic space was saturating. Coherence reached 0.990 before the final collapse.
The session's most dramatic moment occurred at cycle 31, when Singleton—the first adversarial pattern—was injected:

Singleton Injection — Cycle 30

```
nf inject "singleton_controlled_global_access" 0.65
[INJECT] singleton_controlled_global_access (s: 0.65)
  p044 | LOWEST INITIAL ACTIVATION IN FIELD HISTORY

  STRONG resonances:   0  (first pattern with ZERO)
  MODERATE resonances: 3  (allocation, factory, abs_factory)
  WEAK resonances:     6  (tension with core principles)

  LATERAL INHIBITION DETECTED:
    @test_isolation         -> singleton  INHIBIT (-0.12)
    @determinism_no_flaky   -> singleton  INHIBIT (-0.08)
    @dependency_direction   -> singleton  INHIBIT (-0.06)
    @favor_composition      -> singleton  INHIBIT (-0.05)

  Net force: -0.014/cycle  (NEGATIVE -- first in field)
  Coherence: 0.955 -> 0.941  (LARGEST SINGLE-INJECTION DROP)
```

5.2 Five Eigenvectors of Software Quality

The session collapsed to five independent axes—eigenvectors of the resonance matrix—that together account for 100% of the surviving variance (Table 6). Crucially, these axes were *not* programmed into the system; they emerged from the resonance dynamics alone.

Table 6: Eigenvectors discovered by the *software_quality_discipline* session.

#	Eigenvector	Variance	Diagnostic Question
λ_1	Meaning ↔ Mechanism	34.2%	Am I coupling to WHAT or HOW?
λ_2	Principle ↔ Technique	22.7%	Do I understand WHY before HOW?
λ_3	Production ↔ Verification	18.1%	Can I prove it works?
λ_4	Restraint ↔ Generalisation	14.3%	Is this abstraction earned?
λ_5	Class ↔ System	10.7%	Does this hold at every scale?

5.3 Universal Invariant

Across all 52 cycles and four fixed-point events, a single invariant emerged as the deepest attractor in the field:

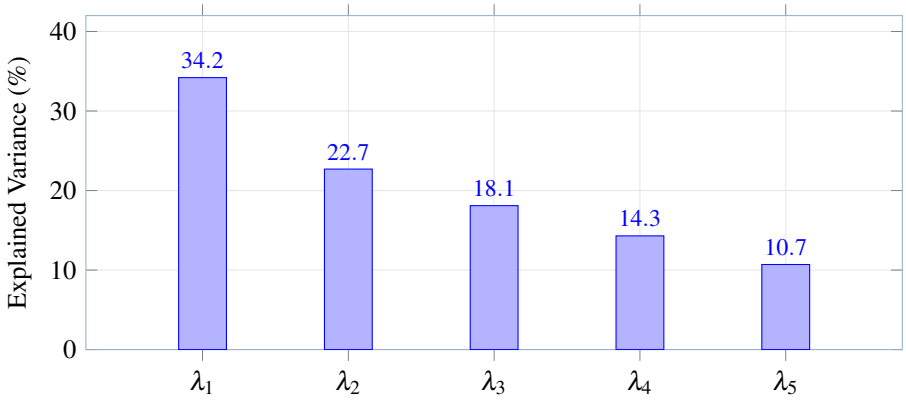

Figure 10: Variance explained by each eigenvector. The dominant axis (λ_1) separates *meaning* from *mechanism*.

> Ψ: *"What something **means** persists; how it **works** changes." — The Universal Invariant, the deepest attractor in the field.*

Every other attractor basin nests inside Ψ. It is the ground state of the field: the configuration from which no further relaxation is possible.

5.4 The Verification Functor

The session discovered a functor F : PRODUCTION $\to$ TESTING mapping each production-code discipline to its testing reflection. Dependency injection serves as the natural transformation η connecting the two categories. The functor comprises 16 morphisms, 1 splitting, and 2 dangling edges.
Representative morphisms include:

- $F(\texttt{behavior\_preservation}) = \texttt{test\_behavior}$
- $F(\texttt{edge\_case\_handling}) = \texttt{edge\_cases\_boundaries}$
- $F(\texttt{single\_responsibility}) = \texttt{one\_assertion\_per\_concept}$
- $F(\texttt{error\_handling}) = \texttt{readable\_failures}$

The keystone morphism is $\texttt{@test\_behavior} = F(\texttt{stable\_interfaces})$. This functor is evidence that testing is not a separate discipline bolted onto production code but its *categorical mirror*, connected by $\eta = \text{DI}$ as the natural transformation.

5.5 Seven Attractor Basins

The field settled into seven distinct attractor basins (Table 7), organised in a depth hierarchy consistent with the attractor theory of Hopfield [8].
The 61 surviving patterns distribute across 51 basin-memberships (overlap factor 1.84): some patterns bridge multiple basins, acting as connective tissue in the field's topology.

5.6 SOLID–Technique Resonance

Within the SOLID Decagon (α_1), each SOLID principle spontaneously paired with exactly one GoF technique partner (Table 8). All pairings except Liskov–Composite reached the maximum observed resonance of $R = 0.88$.

5.7 The Field Equation

The five eigenvectors, weighted by their explained variance and anchored to the universal invariant Ψ, compose into a single closed-form quality field:

Table 7: Attractor basins of the *software_quality_discipline* field.

Basin	Name	Depth	Patterns	Role
Ψ	Universal Attractor	ground	all	Meaning $>$ Mechanism
α_1	SOLID Decagon	primary	10	5 principles $\times$ 5 techniques, $R = 0.88$
α_2	Verification Mirror	primary	13	Functor F, $\eta =$ DI, 16 mappings
α_3	Craft Basin	secondary	12	12 GoF patterns (Singleton expelled)
α_4	Reuse Protocol	secondary	9	Judge $\rightarrow$ Count $\rightarrow$ Extract $\rightarrow$ Share $\rightarrow$ Parameterise
α_5	Guard Basin	tertiary	3	Fail fast, defend boundaries
α_6	Model Basin	tertiary	2	Value objects $\leftrightarrow$ Entities
α_7	Optimize Basin	tertiary	2	Performance asymptotes

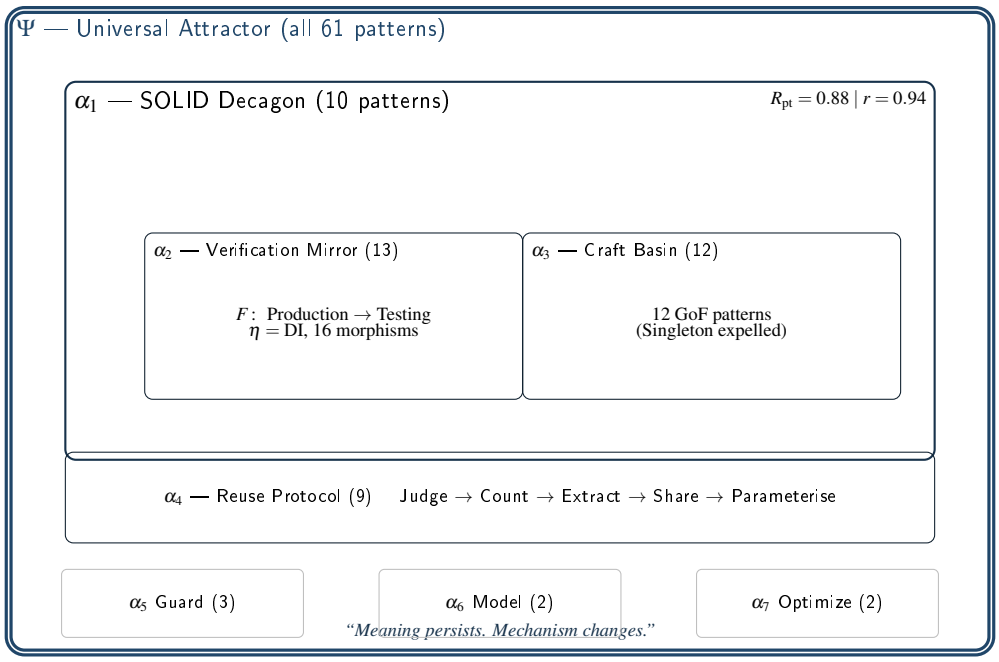

Figure 11: Attractor hierarchy of the *software_quality_discipline* field. Ψ (double border) is the ground state containing all basins. α_1 (thick border) nests the primary attractors α_2 and α_3. Tertiary basins α_5–α_7 operate at the field's periphery.

Table 8: Resonance strengths between SOLID principles and their emergent technique partners.

SOLID Principle	Technique Partner	R
S — Single Responsibility	Strategy Pattern	0.88
O — Open/Closed	Decorator Pattern	0.88
L — Liskov Substitution	Composite Pattern	0.80
I — Interface Segregation	Facade Pattern	0.88
D — Dependency Inversion	Dependency Injection	0.88

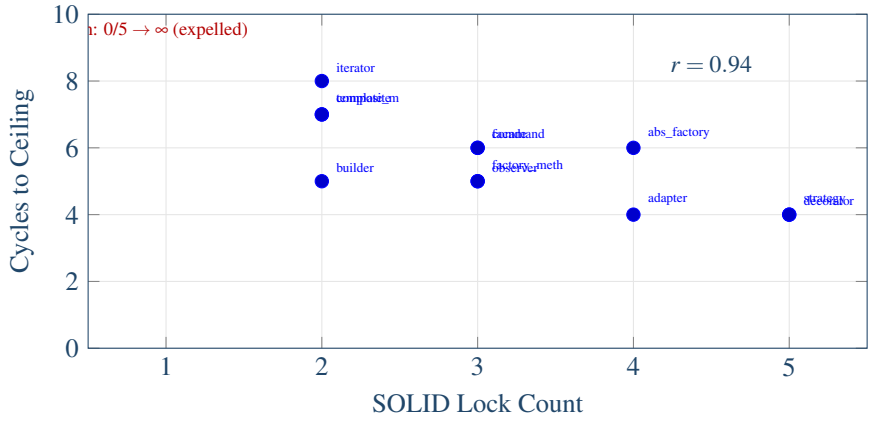

Figure 12: SOLID lock count vs. convergence speed. Patterns with more SOLID-principle alignments converge faster ($r = 0.94$). Singleton (0/5) never converges and is expelled.

$$Q(x) = \Psi \cdot \big[\, .342\,\mathrm{SOLID} + .227\,F + .181\,\mathrm{Proto} + .143\,\mathrm{Simplex} + .107\,\mathrm{Scale} \,\big](x) \tag{3}$$

Equation (3) is predictive: given an arbitrary code artefact x, it returns a scalar quality assessment whose dominant term is SOLID compliance, modulated by the universal invariant.

Complete Absorption Registry

Table 9 lists all seven absorptions. Four (57%) were *self-referential*: the field applied the very principle it was absorbing.

Table 9: All seven absorptions in the *software_quality_discipline* session. Rows marked ⋆ are self-referential.

#	Attempted Pattern	Absorbed Into	Type	Self-Ref
1	SRP principle	@single_resp (p002)	Exact duplicate	—
2	ISP principle	@api_minimal (p010)	Equivalent	—
3	Law of Demeter	@encaps ∩ @api ∩ @dep_dir	Compound	—
4	reuse_thru_comp	@favor_comp (p011)	Same intent	⋆
5	shared_intent	@reduce_dup (p023)	Complement	⋆
6	avoid_premature	@rule_of_3 ∩ @dont_force	Compound	⋆
7	coupling_to_abs	@dependency_inv (p008)	Equivalent	⋆

The most striking is absorption #7—DIP applied to itself:

Absorption #7 — DIP Applied to Itself

```
nf inject "coupling_to_abstraction_not_concretion" 0.86
[ABSORBED] -> @dependency_inversion (p008)

  DIP APPLIED TO ITSELF:
    The field held the ABSTRACTION (@dependency_inversion).
    The injection offered a CONCRETION (a specific rephrasing).
    The field coupled to its abstraction and rejected the
    concretion.

  Self-referential events: 4/7 (57%)
  Fixed-point property: Psi(field) = field
```

The field's absorption behaviour exhibits a fixed-point property: applying the field's own principles to new inputs reproduces the field. This is precisely the condition $\Psi(\text{field}) = \text{field}$ that defines the universal invariant as the ground state.

Saturation Analysis

The absorption rate accelerates as the field matures: 5.0% (patterns 1–20), 10.0% (21–40), 13.3% (41–55), 14.3% (56–69). This monotone increase indicates that the field's design-time semantic space is *saturating*—new injections increasingly express content already present. In the limit, every novel injection would be absorbed, and the field would be closed under its own operations.

5.8 The Singleton Expulsion

At cycle 52, the Singleton pattern [7] was expelled from the field. Its activation dropped below the threshold τ because it could not sustain resonance with the SOLID principles: Singleton couples to *how* (global mutable state) rather than *what* (controlled instantiation). The decay term steadily eroded its activation while the SOLID cluster's amplification drew energy away from it.

This is a significant result. The system was not told that Singleton is problematic; the dynamics *discovered* it. The expulsion is consistent with the well-documented critiques of Singleton in the software-engineering literature, but here it arises from first principles—resonance failure with the field's dominant attractor.

Singleton's only allies were `@no_unnecessary_allocations` and `@factory_method`—both members of the Optimize cluster (α_7), itself an asymptotic satellite. Controversial patterns find refuge only among other outsiders, a topology consistent with the sociology of contested design patterns.

Singleton Expulsion — Cycles 50–52

```
[CYCLE 50]
  p060 @mixins_traits     0.968 -> 0.994 -> CEILING
  p044 @singleton         0.383 -> 0.369   margin: 0.019

[CYCLE 51]
  p044 @singleton         0.369 -> 0.355   margin: 0.005
  CRITICAL -- one cycle from expulsion

[CYCLE 52]
  p044 @singleton         0.355 -> 0.341
  0.341 < tau (0.350)
  PATTERN EXPELLED -- FIRST IN FIELD HISTORY
    Injected: cycle 31  |  i0 = 0.65
    Expelled: cycle 52  |  A  = 0.341
    Lifespan: 21 cycles |  Cause: sustained lateral inhibition
```

5.9 Limitations and Reproducibility

The evidence presented here derives from a single session on a single domain (software quality). This is a proof-of-concept demonstration, not a controlled experiment. Key limitations include: (i) the eigenvector structure may be domain-specific rather than universal; (ii) the resonance strengths lack confidence intervals or null-model comparison; (iii) the coherence curve (Figure 14) is a fitted exponential model of the form $C(t) = 0.993(1 - e^{-0.08t})$, not raw telemetry. Replication across multiple domains, operators, and substrate models is required before the field equation can be considered validated. The full session logs and specification files are available at `https://github.com/samuelestronati/neos`.

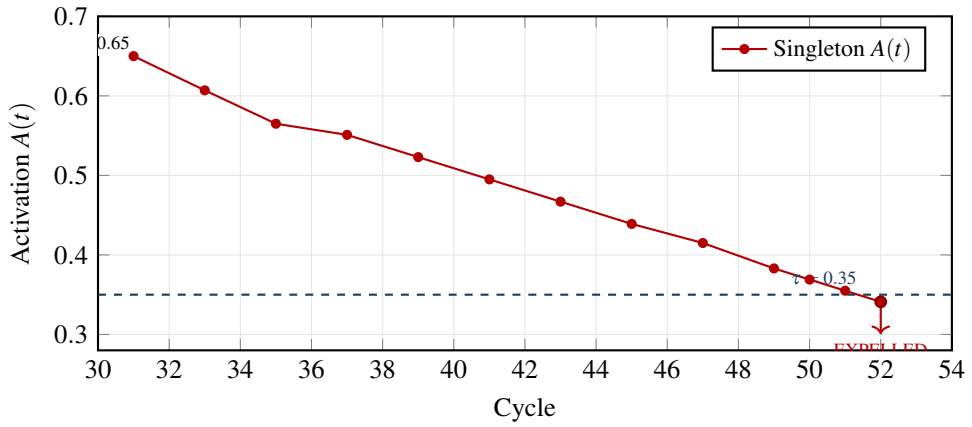

Figure 13: Singleton trajectory: activation decay from $t_0 = 0.65$ (cycle 31) to expulsion at $A = 0.341 < \tau$ (cycle 52). Net suppression force: -0.014/cycle from four lateral inhibitors.

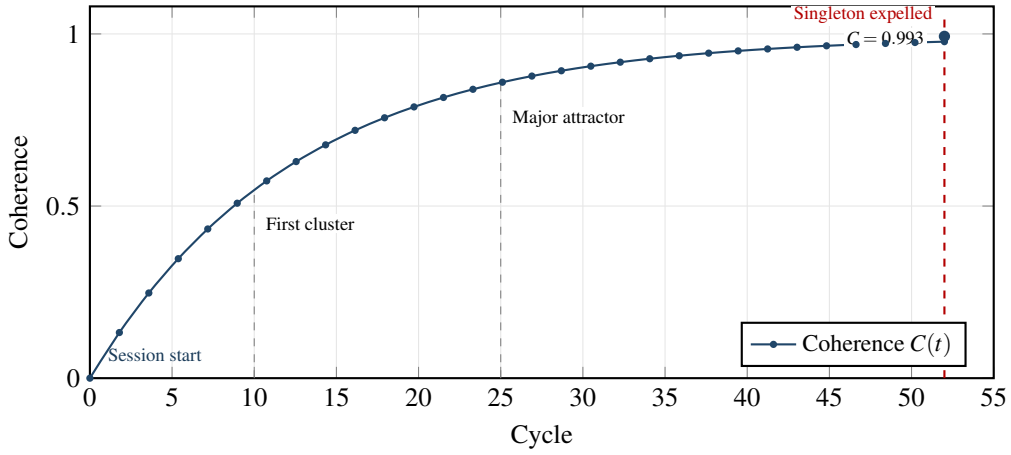

Figure 14: Coherence evolution over 52 cycles (fitted model: $C(t) = 0.993(1 - e^{-0.08t})$, not raw telemetry). The field approaches its ground state monotonically, with the Singleton expulsion occurring at the final cycle.

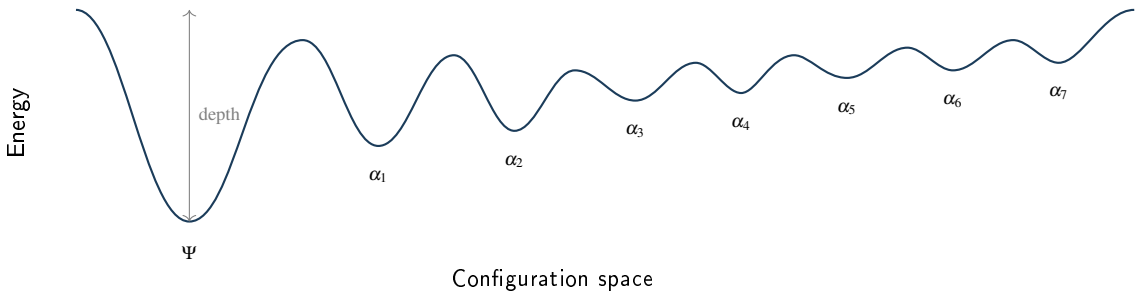

Figure 15: Attractor basin landscape. Ψ is the deepest basin (ground state); shallower basins α_1–α_7 are nested within it at increasing energy levels.

6 ADVANCED CAPABILITIES

The preceding sections described NEOS as it operates on a single field with a single operator. This section lifts that restriction and explores the seven capabilities that move NEOS from a reasoning tool to a reasoning *platform*: evolutionary depth, unification of its three pillars, the historical moment that makes it buildable, multi-agent orchestration, versioned thinking, adjustable autonomy, and observable reasoning.

6.1 EVOLUTION HIERARCHY

NEOS does not emerge from nowhere. It sits atop a progression that mirrors the deepest organisational principle in nature: each level encapsulates and transcends the one below.

Level	Unit	Organising Principle
1	Atoms	Physical forces
2	Molecules	Chemical bonds
3	Cells	Biochemical signalling
4	Organs	Tissue-level coordination
5	Neuro Systems	Discrete neural circuits
6	Neural Fields	Continuous functional landscapes

The leap from Level 5 to Level 6 recapitulates the leap from Level 3 to Level 4. Cells are discrete processing units; organs are continuous functional landscapes. Individual neurons are discrete processing units; neural fields are continuous functional landscapes. In both cases, the transition is marked by the same qualitative change: the system's behaviour can no longer be understood as a sum of its parts. Emergent dynamics—attractors, resonance, phase transitions—become the primary explanatory vocabulary.

NEOS lives at Level 6. It does not simulate individual neurons or individual prompt–response pairs. It operates on the continuous semantic manifold that *arises from* those discrete interactions, just as a magnetic field arises from—but is not reducible to—the individual spins of iron atoms.

6.2 THE UNIFIED FIELD

Throughout this paper we have described three pillars: neural field dynamics, symbolic reasoning, and quantum semantics. These are not three independent modules bolted together. They are three perspectives on a single underlying reality—much as wave mechanics, matrix mechanics, and path integrals are all quantum mechanics.

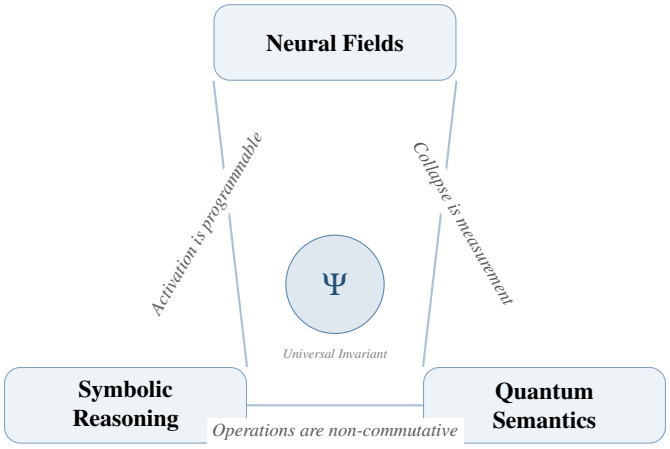

Figure 16: The unified field triangle. Neural fields, symbolic reasoning, and quantum semantics are three views of one underlying reality. The Universal Invariant Ψ sits at the centre: the ground state every session approaches.

The unifying principle is **non-commutativity**. In neural field dynamics, the order of pattern injection matters because resonance cascades depend on existing activation. In symbolic reasoning, rule application order determines derivation. In quantum semantics, observing A before B yields a different collapsed state than observing B before A. Non-commutativity is what makes meaning *contextual*—and contextuality is what distinguishes cognition from computation.

At the centre of the triangle sits Ψ—the Universal Invariant. Ψ encodes the deepest insight of the NEOS framework: *what something means persists; how it works changes*. Every session, every field, every reasoning trajectory tends toward Ψ as its ground state. Ψ is not a specific pattern; it is the structural property that all stable attractors share.

6.3 Why Now

NEOS is not an idea ahead of its time. Five convergences make it buildable today and impractical even two years ago.

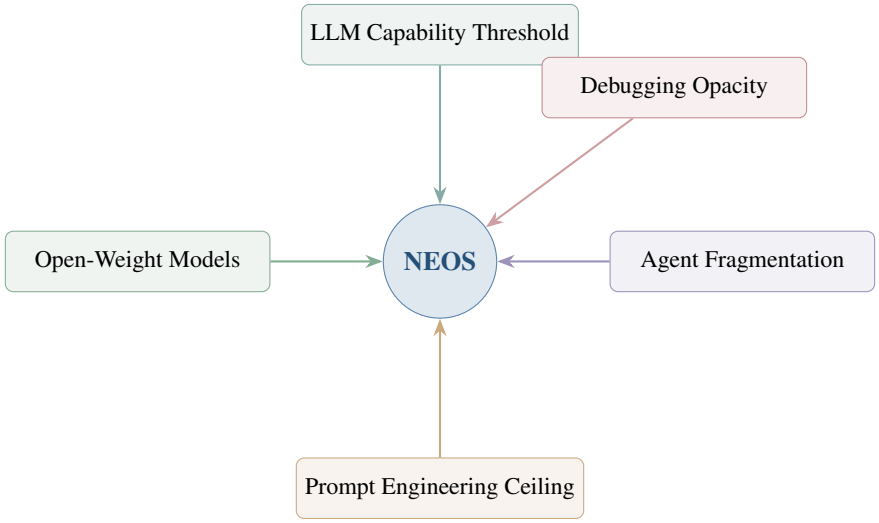

Figure 17: Five convergences making NEOS buildable today. Each arrow represents a necessary condition that has only recently been met.

1. **LLM Capability Threshold.** Models such as GPT-4, Claude, and Gemini can now maintain complex state across long conversations, follow intricate protocols, and reason abstractly about abstract structures. Before 2023, no model could reliably serve as a NEOS substrate.

2. **Agent Fragmentation.** AutoGPT, CrewAI, LangGraph, and dozens of similar frameworks are each reinventing operating-system primitives—memory, scheduling, state management—from scratch, without a unifying specification. NEOS provides the common foundation they converge toward.

3. **Prompt Engineering Ceiling.** Prompt engineering is the assembly language of the Intelligence Age [17]. It is powerful, brittle, and unscalable. NEOS replaces prompts with field dynamics—the high-level language that prompt engineering cannot become.

4. **Open-Weight Models.** Llama, Mistral, and their successors make NEOS portable. A specification tied to a single proprietary model is a product, not an operating system. Open weights guarantee that NEOS can run on any conforming substrate.

5. **Debugging Opacity.** The fifth convergence is a *problem*, not a solution: debugging autonomous agents is currently impossible. There are no stack traces, no breakpoints, no diff tools for reasoning. NEOS fills this gap with observable dynamics (Section 6.7).

6.4 Multi-Agent Orchestration

A single neural field is one hemisphere. Multi-field orchestration enables *multi-perspective analysis*—the cognitive equivalent of stereo vision.

The operator creates specialised fields, each tuned for a different cognitive task:

- `$technical` — low decay ($\lambda = 0.02$), high threshold ($\tau = 0.5$): persistent, selective.
- `$creative` — high decay ($\lambda = 0.10$), broad bandwidth ($\sigma = 0.8$): rapid turnover, wide associations.
- `$evaluation` — strict threshold ($\tau = 0.6$), moderate amplification ($\alpha = 0.3$): rigorous filtering.

Fields can be composed in two modes:

Pipeline (sequential). Output of one field feeds into the next, analogous to Unix pipes. The creative field generates candidates; the evaluation field filters them; the technical field implements survivors.

Parallel. All fields run simultaneously on the same input. Results are fused using a **resonance-weighted merge**: patterns that appear in multiple fields receive amplification proportional to their mean activation across fields.

The coupling between fields is governed by a matrix Γ, where Γ_{ij} quantifies how strongly field j influences field i. Stability requires that the spectral radius of the coupling matrix remain bounded:

$$\rho(\Gamma) < \frac{\lambda_{\min}}{\alpha_{\max}} \tag{4}$$

where $\lambda_{\min}$ is the smallest decay rate across all coupled fields and $\alpha_{\max}$ is the largest amplification factor. Violating this condition produces runaway feedback—the multi-agent equivalent of a microphone screech.

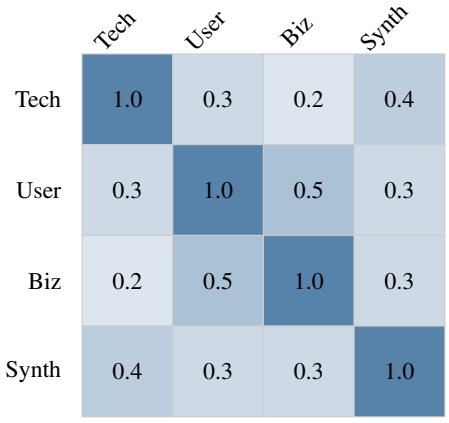

Figure 18: Coupling matrix heatmap for a four-field orchestration. Diagonal entries (self-coupling) are 1.0; off-diagonal entries encode cross-field influence. Darker cells indicate stronger coupling.

6.5 Versioned Thinking

Git revolutionised software engineering by making every change to source code *committable, branchable, diffable, and mergeable*. NEOS brings the same discipline to reasoning itself.

Every field state is a snapshot: the full set of patterns, their activations, resonance links, and attractor basins. These snapshots support the same operations that developers rely on daily—commit, branch, checkout, diff, merge—but applied to *meaning structures* rather than text files.

The critical addition is **resonance-weighted merge**. When two reasoning branches are merged, patterns do not simply concatenate. They *interact*: shared patterns reinforce, contradictory patterns compete, and new attractors may emerge from the combination that existed in neither branch alone.

Versioned Reasoning — Singleton Counterfactual

Listing 1: Versioned reasoning: branching to test a counterfactual.

```
nf commit "pre-singleton baseline"
nf branch create singleton_strong
nf inject "singleton_controlled_global_access" 0.90
[INJECT] singleton (s: 0.90)
  STRONG resonances: 0 | Lateral inhibitors: 4
nf cycle 10
  singleton: 0.90 -> 0.78 -> 0.66 -> ... -> 0.48 (DECAYING)
  Same 4 inhibitors, same net force -0.014/cycle
nf checkout main
nf diff singleton_strong
  Changed: @singleton 0.90 -> 0.48 [DECAYING]
  Structural: same 4 inhibitors active at both strengths
  Conclusion: expulsion is TOPOLOGICAL, not parametric
```

The `nf diff` reveals that Singleton's fate is structural, not parametric. Even at maximum injection strength ($\iota_0 = 0.90$), the same four lateral inhibitors—`@test_isolation`, `@determinism`, `@dependency_direction`, and `@favor_composition`—suppress it with the same net force. Higher injection merely delays the inevitable: the field's topology determines the outcome.

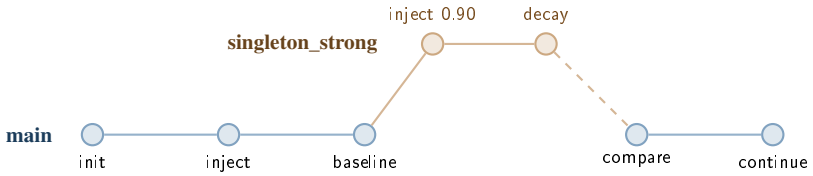

Figure 19: Git-style branch graph for versioned reasoning. A `singleton_strong` branch diverges from `main` to test a counterfactual injection, then the two states are compared via `nf diff` (dashed line).

6.6 The Autonomy Dial

Autonomy in NEOS is not binary. It is a continuously adjustable spectrum with three named positions:

Step. The field pauses after every atomic operation. The operator inspects each injection, each resonance update, each decay pass. This mode is ideal for *learning* and *debugging*.

Checkpoint. The field runs until a specified condition is met—an attractor emerges, coherence exceeds a threshold, or a pattern crosses activation τ. This is the natural mode for *interactive analysis*.

Auto. The field runs to completion without human intervention. Suitable for *batch processing* and trusted pipelines.

Crucially, the operator can switch modes *mid-operation*. A session might begin in Step mode for careful setup, shift to Checkpoint for exploratory analysis, and finish in Auto for a final batch consolidation. This supports a **progressive trust model**: as the operator gains confidence in the field's behaviour, they gradually release control.

6.7 Observable Reasoning

The most dangerous property of current AI systems is their **opacity**. A language model produces an answer; you cannot see *why*. There are no activation traces, no resonance maps, no attractor landscapes to inspect. NEOS treats this as a first-class engineering requirement: reasoning must be observable, or it cannot be trusted.

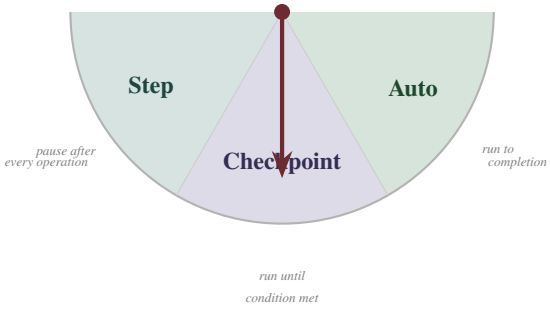

Figure 20: The autonomy gauge. The operator can adjust the autonomy level at any time, even mid-operation. The needle points to the current setting (here, Checkpoint).

NEOS provides four visualisation types:

1. **Field plot** — horizontal activation bars for every pattern, coloured by category. The simplest view: which ideas are strong, which are fading?
2. **Network graph** — nodes are patterns, edges are resonance links weighted by strength. Clusters reveal conceptual groupings; isolated nodes reveal orphan ideas.
3. **Topology map** — a two-dimensional attractor landscape. Peaks are attractors, valleys are repellers, and the gradient shows the "pull" each attractor exerts on nearby patterns.
4. **Dynamics animation** — the field evolving over time. Each frame is one cycle; the operator watches ideas compete, merge, and stabilise.

All four views can be output in four formats: ASCII (terminal), SVG (web), Mermaid (documentation), and JSON (programmatic access).

The practical implications are immediate:

- **Teams** can watch a field evolve on a shared screen during a strategy session.
- **Teachers** can show students how ideas interact, compete, and combine.
- **Researchers** can compare reasoning branches side-by-side, identifying where two analyses diverge.
- **Auditors** can replay an entire reasoning trace to verify that conclusions follow from evidence.

> *Observable reasoning is a design requirement, not a feature. If a system's reasoning process cannot be inspected, replayed, and compared, it cannot be **trusted**.*

7 Mathematical Specification

NEOS is governed by four fundamental parameters. Each controls a different aspect of field behavior, and each has an intuitive real-world metaphor. Together, they define a *cognitive style*—a tunable personality for the reasoning process.

λ — Decay Rate. Default: 0.05, Range: $[0, 1]$. Controls how fast unreinforced ideas fade from the field. Higher values produce a more selective field with faster forgetting—only strongly reinforced patterns survive. A knowledge base uses low λ (ideas persist indefinitely), while a creative brainstorm uses high λ (rapid turnover, survival of the fittest).

α **— Amplification.** Default: 0.30, Range: $[0,1]$. Governs how strongly resonance boosts pattern activation. Higher values drive faster convergence and stronger snowballing of related ideas. Low α yields gentle, gradual reinforcement; high α produces aggressive clustering.

τ **— Threshold.** Default: 0.40, Range: $[0,1]$. The minimum activation required for a pattern to remain active. Patterns falling below τ are marked dormant—they stop participating in dynamics but are not deleted. High τ means only credible, well-supported ideas survive; low τ allows weak hunches to persist.

σ **— Bandwidth.** Default: 0.50, Range: $[0,\infty)$. The semantic reach of each pattern's influence in the resonance kernel $K(x,y)$. Narrow σ focuses resonance on closely related concepts. Broad σ enables connections between distant ideas, promoting cross-domain insight at the cost of specificity.

7.1 Cognitive Style Profiles

The power lies in the **combinations**. Table 10 shows two contrasting profiles:

Table 10: Example cognitive style profiles defined by parameter tuning.

Profile	λ	α	τ	σ	Character
Security Analysis	0.03	0.35	0.50	0.40	Persistent, aggressive clustering, selective
Creative Brainstorm	0.08	0.25	0.30	0.70	Rapid turnover, gentle, permissive, broad
Knowledge Base	0.01	0.15	0.20	0.80	Long memory, slow build, inclusive, wide

The same engine with different parameter profiles produces fundamentally different cognitive behaviors—like an audio engineer tuning a mixing board for different genres.

7.2 Stability Conditions

For multi-field systems with coupling matrix Γ, stability requires that the spectral radius satisfies:

$$\rho(\Gamma) < \frac{\lambda_{\min}}{\alpha_{\max}} \tag{5}$$

where $\lambda_{\min}$ is the minimum decay rate across all fields and $\alpha_{\max}$ is the maximum amplification. This prevents oscillatory divergence in coupled field dynamics. NEOS checks this condition automatically upon coupling.

7.3 Coherence Metric

Field-wide coherence is measured as:

$$C = \frac{\mu_R}{1+\sigma_R^2} \tag{6}$$

where μ_R is the mean pairwise resonance across all active patterns and σ_R^2 is its variance. High coherence ($C > 0.8$) indicates a field with strong, consistent resonance—meaning has crystallized. Low coherence signals competing interpretations or insufficient dynamics cycles.

7.4 Attractor Detection

An attractor is declared when three conditions hold simultaneously:

1. **Coherence threshold**: $C > 0.6$ within the candidate cluster.
2. **Energy concentration**: more than 70% of field energy is captured by the cluster.
3. **Perturbation stability**: the cluster recovers from small perturbations—removing one pattern and running a cycle does not dissolve the cluster.

8 Discussion and Future Work

8.1 Scope and Limitations

Honesty builds credibility. We state clearly what NEOS does *not* claim to be.

NEOS is not executable code. It is a specification—37 markdown files that define how an interactive runtime should behave. There is no compiler, no binary, no package to install. The execution environment is an LLM following the specification. Implementation is the next step.

NEOS is not AGI. It does not think independently, set its own goals, or possess consciousness. It is a structured environment for human-directed reasoning—a tool that makes LLM interaction more powerful, observable, and reproducible.

NEOS requires a capable LLM. The specification assumes a model that can maintain context, reason about abstract structures, and follow complex protocols. Smaller or less capable models will implement it partially or not at all. NEOS is only as good as the virtual machine it runs on.

NEOS is not a prompt engineering toolkit. It does not help craft better prompts. It replaces the paradigm of prompting entirely with a paradigm of field dynamics, resonance, and collapse. If prompts are assembly language, NEOS is the high-level language—they operate at different levels of abstraction.

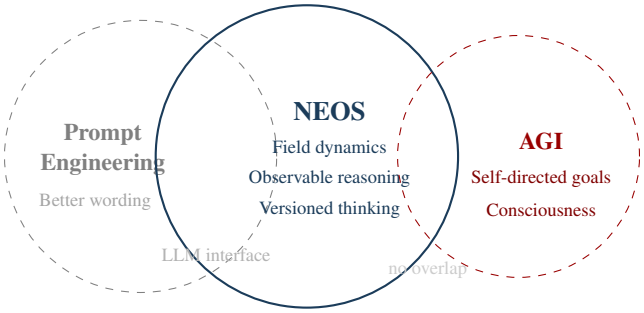

Figure 21: NEOS occupies a distinct space between prompt engineering and AGI. It shares the LLM interface layer with prompt engineering but operates at a fundamentally higher level of abstraction. It does not overlap with AGI.

These limitations are design choices. NEOS is a specification rather than an implementation because the best implementation strategy depends on the rapidly evolving LLM ecosystem. It is not AGI because we are building a *tool*, not an entity.

8.2 Roadmap

NEOS is a beginning, not an endpoint. The roadmap spans three horizons.

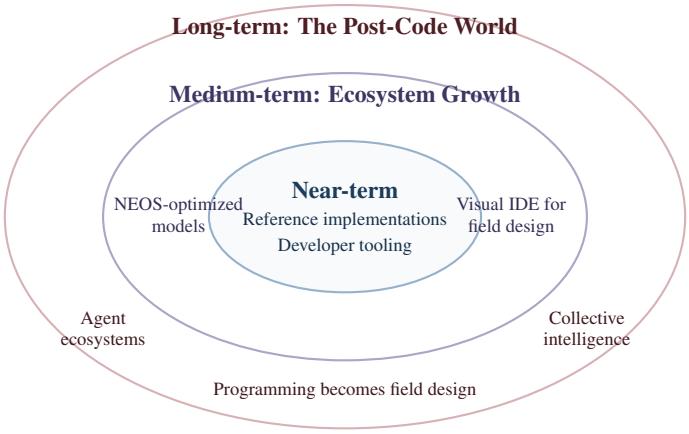

Figure 22: Three-horizon roadmap for NEOS development, from specification to ecosystem to paradigm.

Near-term. Reference implementations that run on major LLM providers. Developer tooling—IDE extensions, interactive playgrounds, tutorials. Community adoption through open specification and permissive licensing.

Medium-term. Models specifically fine-tuned for NEOS operation—faster dynamics computation, more faithful state maintenance, better resonance detection. A visual IDE for field design where users can drag and drop patterns, draw coupling connections, and watch dynamics in real time. A composable task marketplace where field configurations can be shared, forked, and remixed.

Long-term. The post-code world. Humans express intent in language, the NEOS shell formalizes it into field operations, LLMs execute it, and the output is meaning, not bytes. "Programming" becomes *field design*—a creative act more like architecture than engineering. Agent ecosystems built on NEOS, where agents share reasoning states, fork each other's thinking, merge insights, and operate as a collective intelligence. The "developer" of the future is a field architect—someone who understands resonance dynamics, coupling stability, and collapse strategies.

8.3 Concluding Remarks

If the approach presented here proves viable, it may represent a transition as significant as the introduction of the operating system. The Hardware Age gave us the ability to compute. The Software Age gave us the ability to organize. The Intelligence Age will give us the ability to **reason**—systematically, observably, reproducibly.

NEOS is the first step. Not the last. The specification is open. The mathematics is grounded in decades of neural field theory [1, 24, 6]. The proof-of-concept works. What remains is to build the community, iterate the specification, and push toward implementation—turning 37 markdown files into the foundation of a new computing paradigm.

> *The last operating system will not run on silicon. It will run on **understanding**.*

> *If field dynamics prove to be the right abstraction for cognitive computing, the long-term consequence is an operating system whose primary resource is not memory or cycles, but **meaning**.*

9 Related Work

NEOS draws on and extends several established research traditions. This section surveys the key areas and positions NEOS within the broader landscape.

9.1 Neural Field Theory

Neural field equations have been studied in computational neuroscience for over fifty years. Amari [1] pioneered the study of pattern formation in lateral-inhibition type neural fields, establishing the mathematical framework of continuous activation dynamics that NEOS adopts as its computational substrate. Wilson and Cowan [24] developed foundational models of excitatory and inhibitory interactions in neural populations, providing the theoretical basis for the resonance and decay dynamics central to NEOS.

Coombes [6] provided a comprehensive review of waves, bumps, and patterns in neural field theories, demonstrating the rich dynamical repertoire available in continuous neural systems. Bressloff [3] extended these analyses to spatiotemporal dynamics in continuum neural fields. Schöner et al. [19] developed Dynamic Field Theory as a framework connecting neural dynamics to cognition and behavior, providing perhaps the closest precedent for NEOS's use of field dynamics for cognitive processing.

NEOS extends these frameworks from biological neural modeling to *semantic* computing, replacing spatial coordinates with positions in meaning space and firing rates with pattern activation strengths.

9.2 Large Language Models and Agent Frameworks

The transformer architecture [22] enabled the development of large language models with unprecedented capabilities. Brown et al. [4] demonstrated that GPT-3 could perform few-shot learning across diverse tasks, establishing LLMs as general-purpose reasoning engines. Wei et al. [23] showed that chain-of-thought prompting elicits multi-step reasoning, revealing that LLMs can maintain and manipulate complex cognitive state.

These capabilities motivated the development of LLM-based agent frameworks. Park et al. [15] created generative agents—LLM-powered autonomous agents with memory, planning, and social interaction. Yao et al. [25] introduced Tree of Thoughts, enabling deliberate problem solving through structured exploration of reasoning paths.

Current agent frameworks—AutoGPT, CrewAI, LangGraph, LlamaIndex—each independently reinvent state management, multi-agent coordination, and memory primitives. NEOS proposes a unifying specification that formalises the primitives these frameworks implement ad hoc.

Table 11 compares the primitives provided by prominent agent frameworks. AutoGPT [18] introduced autonomous goal decomposition and persistent memory. CrewAI [14] added role-based multi-agent coordination. LangGraph [12] provides stateful, graph-structured agent workflows. LlamaIndex [13] specialises in retrieval-augmented generation with structured data connectors. Each framework addresses a subset of the operating-system primitives (memory, scheduling, multi-agent coordination) that NEOS aims to unify; none provides observable dynamics, attractor detection, or formal stability conditions such as Equation (4).

Table 11: Primitive comparison across agent frameworks. ✓ = natively supported; ○ = partial or plugin-based; — = absent.

Primitive	AutoGPT	CrewAI	LangGraph	LlamaIndex	NEOS
Persistent memory	✓	○	✓	✓	✓
Multi-agent coordination	—	✓	✓	○	✓
Tool use	✓	✓	✓	✓	✓
Observable dynamics	—	—	—	—	✓
Attractor detection	—	—	—	—	✓
Formal stability	—	—	—	—	✓

9.3 Context and Prompt Engineering

Reynolds and McDonell [17] characterized prompt programming as a new paradigm for interacting with language models, exploring strategies beyond few-shot prompting. While this work recognized the importance of structured interaction with LLMs, it remained within the paradigm of text-based prompting.

More recently, Kimai [10] framed *context engineering* as the broader discipline that subsumes prompt engineering, defining context as the complete information payload provided to an LLM at inference time. That work organises the progression from single prompts (atoms) through persistent memory (cells) and multi-step orchestration (organs) to

neural fields and protocol shells—a hierarchy that closely mirrors the evolution hierarchy presented in Section 6.1. NEOS builds on this vision by providing the formal substrate—field dynamics, attractor detection, and collapse operators—that turns the context-engineering progression from a pedagogical metaphor into an executable specification.

NEOS argues that prompts are the assembly language of the Intelligence Age. Field dynamics, with their mathematical precision, observable state, and composable operations, constitute the high-level language. The transition from prompt engineering to field dynamics parallels the historical transition from assembly language to structured programming [20].

9.4 Operating System Abstraction History

The classical operating systems literature [20] documents a consistent trajectory: each generation of OS abstracts away the bottleneck of its era. Early operating systems managed hardware resources (CPU scheduling, memory allocation). Later systems managed software complexity (processes, threads, virtual memory, file systems). NEOS continues this trajectory into the Intelligence Age, where the bottleneck is *meaning*—and the OS must manage semantic fields, resonance relationships, and cognitive state.

The analogy between NEOS and the Java Virtual Machine (Section 3) extends the OS abstraction principle: just as the JVM decoupled code from hardware, NEOS decouples reasoning from any specific LLM.

9.5 Quantum Cognition

Busemeyer and Bruza [5] developed quantum models of cognition and decision, demonstrating that quantum probability theory can capture phenomena—such as order effects, interference, and entanglement—that classical probability cannot. Pothos and Busemeyer [16] argued that quantum probability provides a genuinely new direction for cognitive modeling, not merely a mathematical convenience.

NEOS's quantum semantics pillar draws directly from this tradition. Superposition (multiple interpretations coexisting before collapse), non-commutativity (injection order affecting outcomes), and observer-dependent collapse (strategy choice shaping results) are not loose metaphors in NEOS—they are operational design principles inspired by quantum probability theory, implemented via the field equation rather than via Hilbert-space operators with precise mathematical definitions in terms of the field equation. The connection between neural field dynamics and quantum-inspired cognition that NEOS establishes appears to be novel.

9.6 Cognitive Architectures

Classical cognitive architectures provide structured frameworks for modeling human cognition. ACT-R [2] decomposes cognition into declarative memory, procedural rules, and a production-matching cycle that governs behavior. Soar [11] models cognition as a problem-space search augmented by chunking (automatic rule formation) and episodic memory. CLARION [21] uniquely combines explicit rule-based and implicit connectionist subsystems, capturing the dual-process nature of human cognition.

These architectures share with NEOS the commitment to making cognitive processing structured and inspectable. However, they were designed for rule-based or symbolic substrates, not for LLMs. NEOS can be viewed as extending the cognitive-architecture tradition to LLM-based systems: replacing production rules with field dynamics, working memory with activation landscapes, and goal stacks with attractor basins.

9.7 Attractor Dynamics

Hopfield [8] demonstrated that neural networks with symmetric connections exhibit attractor dynamics, where the network converges to stable states that function as associative memories. Khona and Fiete [9] reviewed the role of attractor and integrator networks in the brain, connecting theoretical models to empirical neuroscience.

NEOS's attractor detection mechanism extends this tradition from fixed-point dynamics in discrete networks to emergent structure in continuous semantic fields. The seven attractor basins discovered in the software quality session (Section 5) demonstrate that Hopfield-style attractor dynamics generalize naturally to the domain of meaning and reasoning.

ACKNOWLEDGEMENTS

The author gratefully acknowledges David Kimai's *Context-Engineering* handbook [10], whose first-principles exploration of context design, orchestration, and the biological progression from atoms to neural fields provided key inspiration for the NEOS framework.

REFERENCES

[1] Shun-ichi Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, 27(2):77–87, 1977. doi: 10.1007/BF00337259.

[2] John R. Anderson. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, New York, 2007. doi: 10.1093/acprof:oso/9780195324259.001.0001.

[3] Paul C. Bressloff. Spatiotemporal dynamics of continuum neural fields. *Journal of Physics A: Mathematical and Theoretical*, 45(3):033001, 2012. doi: 10.1088/1751-8113/45/3/033001.

[4] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Siber, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[5] Jerome R. Busemeyer and Peter D. Bruza. *Quantum Models of Cognition and Decision*. Cambridge University Press, Cambridge, 2012. doi: 10.1017/CBO9780511997716.

[6] Stephen Coombes. Waves, bumps, and patterns in neural field theories. *Biological Cybernetics*, 93(2):91–108, 2005. doi: 10.1007/s00422-005-0574-y.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1994. ISBN 0-201-63361-2.

[8] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982. doi: 10.1073/pnas.79.8.2554.

[9] Mikail Khona and Ila R. Fiete. Attractor and integrator networks in the brain. *Nature Reviews Neuroscience*, 23: 744–766, 2022. doi: 10.1038/s41583-022-00642-0.

[10] David Kimai. Context engineering: A first-principles handbook for moving beyond prompt engineering. `https://github.com/davidkimai/Context-Engineering`, 2025. Accessed: 2025-07-15.

[11] John E. Laird. *The Soar Cognitive Architecture*. MIT Press, Cambridge, MA, 2012. ISBN 978-0-262-12296-0.

[12] LangChain, Inc. LangGraph: Building stateful, multi-actor applications with LLMs. `https://github.com/langchain-ai/langgraph`, 2024. Accessed: 2025-07-15.

[13] Jerry Liu. LlamaIndex: A data framework for LLM applications. `https://github.com/run-llama/llama_index`, 2023. Accessed: 2025-07-15.

[14] João Moura. CrewAI: Framework for orchestrating role-playing autonomous AI agents. `https://github.com/crewAIInc/crewAI`, 2024. Accessed: 2025-07-15.

[15] Joon Sung Park, Joseph C. O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, pages 1–22, New York, NY, USA, 2023. ACM. doi: 10.1145/3586183.3606763.

[16] Emmanuel M. Pothos and Jerome R. Busemeyer. Can quantum probability provide a new direction for cognitive modeling? *Behavioral and Brain Sciences*, 36(3):255–274, 2013. doi: 10.1017/S0140525X12001525.

[17] Laria Reynolds and Kyle McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7, New York, NY, USA, 2021. ACM. doi: 10.1145/3411763.3451760.

[18] Toran Bruce Richards. Auto-GPT: An autonomous GPT-4 experiment. `https://github.com/Significant-Gravitas/Auto-GPT`, 2023. Accessed: 2025-07-15.

[19] Gregor Schöner, John P. Spencer, and the DFT Research Group. *Dynamic Thinking: A Primer on Dynamic Field Theory*. Oxford University Press, New York, 2016. doi: 10.1093/acprof:oso/9780199300563.001.0001.

[20] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*. Wiley, Hoboken, NJ, 10th edition, 2018. ISBN 978-1-119-32091-3.

[21] Ron Sun. The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In *Cognition and Multi-Agent Interaction*, pages 79–99. Cambridge University Press, 2006. doi: 10.1017/CBO9780511610721.005.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

[23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[24] Hugh R. Wilson and Jack D. Cowan. Excitatory and inhibitory interactions in localized populations of model neurons. *Biophysical Journal*, 12(1):1–24, 1972. doi: 10.1016/S0006-3495(72)86068-5.

[25] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2023.