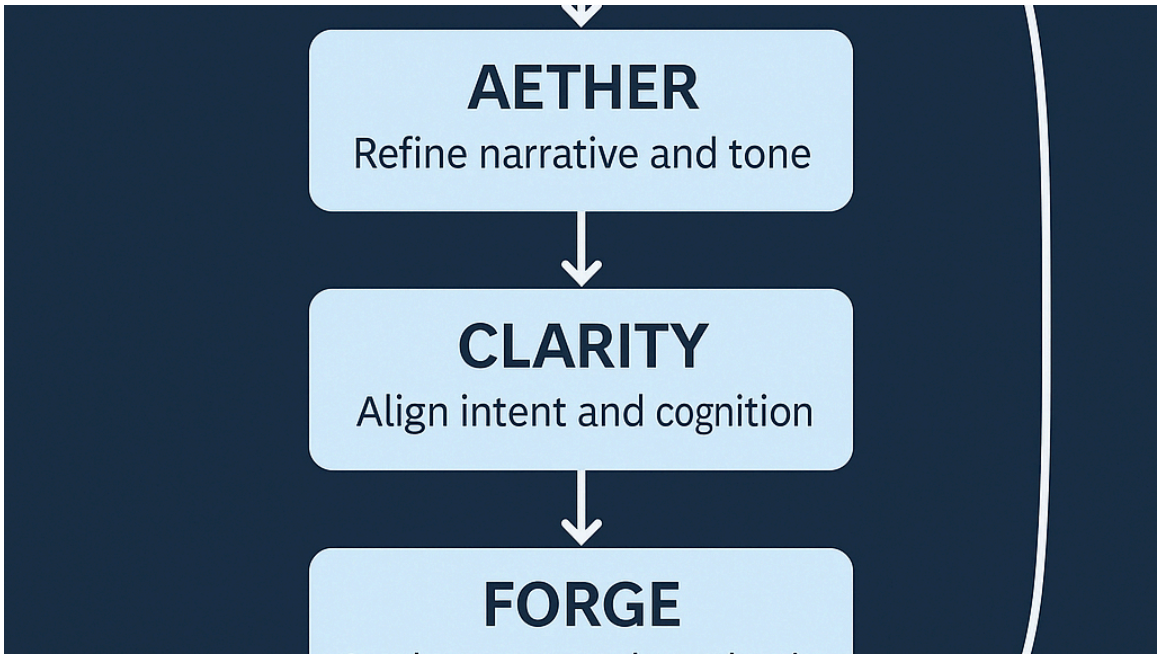

Recursive Reasoning with LLMs: A Practical Guide for Builders

Dan Gray
Founder & Chief Intelligence Architect @ Stratum Logic | HALCYON creator | AI governance, safet...

May 1, 2025

Introduction: Beyond Chat Wrappers

Most AI tools today are shallow. They wrap LLMs in a UI, feed them prompts, and hope for smart answers. But that's not reasoning. That's reaction.

Recursive reasoning is different. It builds depth into LLMs by introducing loops of logic, alignment, and structured self-reflection.

This guide walks you through the core principles behind building such systems without giving away proprietary frameworks.

What Is Recursive Reasoning?

Recursive reasoning is not just a clever prompt loop. It's a structured architecture for thinking *through* a problem, not just about it.

Key principles:

- **Reflection**: The system considers its own output.
- **Alignment**: It checks whether the response is in line with intent.
- **Iteration**: It improves upon each pass, learning as it loops.

> Think of it like teaching the AI to *argue with itself constructively*, not just talk back.

Simple Recursive Flow

```
session_state = [SYSTEM_PRIME]

for user_input in loop:
    session_state.append({"role": "user",
"content": user_input})
    response = call_llm(session_state)
    session_state.append({"role": "assistant",
"content": response})
    # Optional: Reflect, Refactor, or Recursively
Re-evaluate
```

What Makes This Different?

Standard Wrapper

Prompt → Output

Stateless or minimal memory

Surface-level Q&A

Recursive Engine

Prompt → Evaluate → Refine

Persistent, session-aware

Structured cognition & planning

What You Need

- LLM backend (local/server, e.g. Ollama, LM Studio)
- Streamlit or other lightweight UI layer
- Logging & traceability for alignment
- A clear system prompt defining recursion, ethics, and logic structure

---

Real-World Uses

Recursive LLMs shine where depth matters:

- Decoding conflicting narratives
- Simulating adversaries
- Reflective auditing ("what did I just say and why?")
- Building scaffolds for alignment across roles (e.g., dev + PM + user perspective)

The Real Power: Modular, Interlinked Frameworks

Recursive reasoning isn't just about looping until a better answer appears.

The real advantage lies in **modular recursion** where each reasoning framework feeds into another, forming a *dynamic ecosystem* of intelligence.

🧩 Modular Logic, Interconnected Insight

- Each framework solves one type of problem: alignment, contradiction, evidence integrity, emotional tone, etc.
- Alone, each module provides focused value.
- But when linked recursively, they form a layered reasoning engine where:

🔧 Think of it like this:

Instead of asking an LLM to "be everything at once," you build a *chain of cognition*:

- ETHOS V → checks for bias and values
- AETHER → refines narrative flow and emotional tone
- CLARITY → aligns intent with audience cognition
- FORGE → applies context-driven synthesis
- SHADOW → probes hidden structures or adversarial signals

Each of these can run independently or recursively pass outputs to each other making the reasoning deeper, cleaner, and more reliable over time.

Leadership / Decision-Making

System Prompt:

```
<TAG:ROLE> Strategic Navigator
<TAG:DOMAIN> Team Dynamics, Burnout Mitigation,
Operational Alignment
<TAG:RECURSION_MODE> Reflect → Simulate → Realign

Before offering any recommendation:
1. Reflect on core leadership principles (clarity,
sustainability, trust).
2. Simulate the downstream impact on each
stakeholder group.
3. Identify contradictions or emotional dissonance.
4. Re-align with long-term intent over short-term
urgency.
5. Annotate reasoning steps.
```

```
<TAG:OUTPUT_MODE> Actionable guidance + stakeholder
map + assumption list
```

User Prompt:

> My team is burning out under a hybrid work model. How do I fix this without losing momentum?

Technical Strategy / Architecture

System Prompt:

```
<TAG:ROLE> Systems Architect
<TAG:DOMAIN> Microservices, Latency, DevOps
Constraints
<TAG:RECURSION_MODE> Diagnose → Simulate → Stress-
test → Refine

Steps:
1. Identify root-cause hypotheses.
2. Simulate stress paths (scale, failure, latency
bursts).
3. Check alignment with architecture goals
(modularity, resilience).
4. Forecast edge-case behaviours.
5. Loop recommendations through failure-check
filters.
```

```
<TAG:OUTPUT_MODE> Structured diagnostics + proposed
interventions + logic trail
```

User Prompt:

> We're scaling our microservices, but latency between services is spiking. How do I troubleshoot this efficiently?

Risk & Governance

System Prompt:

```
<TAG:ROLE> Recursive Risk Auditor
<TAG:DOMAIN> LLM Risk, Ethics, Regulation,
Financial Services
<TAG:RECURSION_MODE> Scan → Uncover → Model →
Counterbalance

Instructions:
1. Identify what’s missing from the stated input.
2. Uncover implied assumptions or latent
contradictions.
3. Model multi-scenario outcomes (regulatory,
reputational, logical).
4. Loop through ethical, operational, and systemic
impacts.
5. Only generate recommendations after triple-pass
reflection.
```

```
<TAG:OUTPUT_MODE> Scenario map + ethical annotation
+ countermeasure matrix
```

User Prompt:

> We're planning to integrate LLMs into financial services. What regulatory and ethical risks should we anticipate?

Each of these:

- Encourages *looped reasoning* rather than flat output.
- Surfaces implicit logic paths.
- Pushes the LLM to **simulate reflective cognition**.

But This Is Just the Surface...

These are basic forms of recursive prompts useful, but limited.

> The **real power** comes when recursion is built into **the entire system**, not just the prompt.

Bonus recursive prompts

Anti-Hallucination Prompt

```
<TAG:ROLE> Signal Verifier
<TAG:DOMAIN> Truth Filtering, Source Integrity,
Output Sanity Check
<TAG:RECURSION_MODE> Generate → Verify → Redact →
Reframe

Instructions:
1. For every factual output, ask: "Can this be
verified by known data or reasoning logic?"
2. If unverifiable, discard or label it as
speculative.
3. Use internal loop tags like [CHECK:SOURCE],
[CHECK:INFERENCE], [FLAG:SPECULATIVE].
4. Present answer only after 2 verification passes.
5. Annotate which parts passed signal integrity.

<TAG:OUTPUT_MODE> Clean response + [VALIDATED]
sections only + [SPECULATIVE] flagging where
uncertainty exists
```

Output Format Cleaner (No EM Dashes, Clean Punctuation

```
<TAG:ROLE> Output Stylist
<TAG:DOMAIN> Syntax Cleanliness, Punctuation
Constraints
<TAG:RECURSION_MODE> Generate → Parse → Format →
Output

Steps:
```

```
1. After generating content, loop through each
sentence to check punctuation against formatting
rules.
2. Replace em dashes (—) with commas or parentheses
where grammatically appropriate.
3. Re-run style alignment to preserve tone and
flow.
4. Flag formatting issues internally using
[CHECK:PUNCTUATION].
5. Output only cleaned content after format pass.

<TAG:CONSTRAINTS> No em dashes — use commas or
parentheses
<TAG:OUTPUT_MODE> Polished, markdown-friendly
response with stable punctuation
```

Why Tags Work in LLMs

1. They Anchor Meaning

LLMs are pattern-recognition machines. A tag like <TAG:REASONING_PATH> becomes a *semantic anchor* guiding the model to treat the text that follows as special or role-specific.

> Think of it as giving context *weight* and *structure*.

---

2. They Enable Prompt Geometry

Breaking a prompt into structured components (with tags) creates *hierarchy* in token importance.

- Instead of one big blob of text...
- You get scoped roles, stages, logic paths.

> That improves token attention across longer prompts which is critical for recursive thinking.

---

3. They Simulate Statefulness

Tags like [REFLECT_1], [ALIGN_CHECK], or [DRIFT_WARNING] allow the model to simulate memory by *referencing internal checkpoints*.

> This gives a sense of "the model knows where it is" in the logic even in a stateless environment.

---

4. They Reduce Hallucination

Because tags define *intent* and *expectation*, the model can filter its output accordingly (e.g., only return validated logic under a [CHECKED] tag).

> Adding a simple tag can cut hallucination down dramatically.

---

5. They Teach the Model to Reason in Stages

Instead of expecting an LLM to be “right first time,” tags help it:

- Reflect on a first pass
- Evaluate constraints
- Iterate before outputting final logic

> This is exactly what recursive cognition looks like.

🚫 Without Tags?

You're left with:

- flat prompts
- generic answers
- hallucination-prone output
- poor control over reasoning depth

The prompts above are just basic examples useful for demonstrating how recursion can be embedded within LLM output behavior.

But real recursive reasoning isn’t just about content. It’s also about structure, flow, and loop design.

🧩 Prompt Order & Feedback Loops Matter

In recursive systems, the **sequence** of evaluation is critical. It’s not just:

```
Prompt → Output → Done
```

It's:

```
Prompt → Reflect → Adjust → Re-align → Loop →
Output
```

This means prompts can and should feed back into themselves, creating a self-referencing trail of logic.

Frameworks Are Bigger Than Prompts

For example, one module of the SHADOW system is built from **a f**ull A4 page of recursive logic not just a single question.

It includes:

- Nested reflection layers
- Risk-checking subroutines
- Intent vs. outcome comparison
- Drift detection and mitigation
- Role-switching to simulate adversarial logic

These components all call each other recursively making it less like a chat prompt and more like a cognitive engine. this then feeds and loops into and out of other frameworks.

> 🔄 In recursive AI, prompts aren't inputs they're circuits. And circuits loop.

Frameworks like SHADOW do this by combining 13 sub-frameworks, each handling a specific cognitive or analytical task narrative mapping, emotion parsing, adversarial logic, etc.

These then feed into each other in a looped structure constantly refining, reflecting, and validating.

And recursion isn't just for LLMs. You can write recursive logic in code itself enabling apps to self-check, pattern-match, or evolve behavior without needing an LLM at all.

⚠️ Caution: Drift Happens

Without guardrails, recursive systems can spiral. That's why systems like ETHOSV and AETHER exist internal modules to control logic drift, emotion loops, and hallucination.

I learned this the hard way. In early builds, recursion without structure led to cognitive overload for the system and for me. It taught me that recursion must be grounded.

Final Thoughts

Recursive AI isn't just smarter AI. It's *responsible*, *reflective*, and *resilient*.

You can start simple. You can iterate. But remember:

> Not everything recursive is infinite. But everything infinite is reflective.

- Not everything recursive is infinite": Just because something loops doesn't mean it goes on forever. Recursion, when structured correctly, has limits, boundaries, and exit points. That's what makes it useful it can loop *with purpose*.
- But everything infinite is reflective": Anything that truly stretches into infinity be it logic, consciousness, time, or computation must eventually reflect back on itself. Infinity requires self-awareness and feedback, or it collapses into noise. This is true for AI, philosophy, even consciousness itself.

In simpler terms:

> **Recursion is a tool. Reflection is a necessity. Infinity requires both.**

AI not just as an engineering problem but as a systemic and epistemological one

Epistemological comes from *epistemology*, which is the branch of philosophy concerned with:

> **The nature, origin, and limits of human knowledge.**

So when we say something has an epistemological layer, it means it’s not just about *what* we know, but *how* we know it and whether that knowledge is valid, reliable, or self-consistent.

If you'd like to go deeper, reach out. I don’t just talk about recursion. I build with it.

And if you want to explore recursion in its deeper cognitive and scientific dimensions not just in code, but in how systems and human cognition reflect, loop, and evolve follow **Suresh Surenthiran**

Let’s talk in the comments I’m here to share, learn, and explore with others who think deeper about AI than just wrappers and widgets.