FATI

Fractal Adaptive Teleonomic Intelligence

Full Specification — Version 1.0

AL_78 | Independent Researcher | June 2026

In collaboration with Claude (Anthropic), ChatGPT (OpenAI), DeepSeek

Important Note

This document represents a conceptual work. Many theoretical statements (especially regarding convergence and stability) are working hypotheses, not proven theorems. The author is an independent researcher, not affiliated with any institution, and does not seek funding. This is an engineering concept, not a formal scientific paper; it contains no proofs, theorems, or experimental results, but rather an architectural blueprint intended to invite discussion and collaboration. The core theoretical novelty — the asymmetric hysteresis kernel — remains hypothetical and has no ready implementation. The MVP roadmap and feasibility table (Section 6) clearly separate components that can be built today from those that require further research.

Contents

Abstract

This document presents FATI (Fractal Adaptive Teleonomic Intelligence) — a comprehensive conceptual architecture for a new class of intelligent systems. FATI is defined as a non-linear, open dynamical system with variable state dimension, where thinking is modeled as a trajectory in a deformable phase manifold. Key hypothetical properties include: non-Markov memory (asymmetric hysteresis kernel), telonomy (minimization of cognitive incongruence), fractal cognitive space, and a dynamic possibility space.

The theoretical framework is mapped to a production-ready engineering stack based on current technologies: Mixture-of-Experts with NoisyTopKRouter, three-tier memory, world model, and meta-controller. Feasibility is demonstrated through detailed engineering blueprints, scaling parameters for up to 256 GPUs, and an MVP roadmap.

This is a first-level conceptual blueprint; many theoretical claims remain hypothetical and require future formalization.

Introduction

Modern large language models have achieved remarkable success in pattern recognition and text generation. However, their fundamental limitation — the Markov property (dependence only on the current context window) — deprives them of true memory and the ability for long-term thinking trajectories.

FATI offers an alternative approach: thinking as a trajectory in a deformable phase manifold. The system features non-Markov memory, variable state dimension, and intrinsic telonomy — goal-directedness without a fixed goal.

The document is organized as follows: **Theory of FATI** (Section 1), **Physical Layer and Dynamical Core** (Section 2), **Mathematical Formalization** (Section 3), **Engineering Parameters** (Section 4), **ML Architecture** (Section 5), **Production Routing** (Section 8), **Deep System Engineering** (Section 9), and the final **MVP Roadmap** (Section 12).

Key definition:

> *"Intelligence is not a computation function but the evolution of the geometry of meaning space under the influence of memory, nonlinearity, and self-reference, where 'thinking' is a trajectory in a deformable phase manifold."*

Researcher's Limitations and Scope

This specification is the work of an independent researcher. All sanity checks were performed on available consumer hardware (mobile phone + Google Colab free tier, Phi-2 2.7B on T4 GPU). The architecture is designed to scale up to 256 GPUs (see Section 4.4) and to be implemented by a team with access to institutional compute resources. The full validation of components such as the asymmetric hysteresis kernel, world model, meta-controller, and self-modification layer requires resources beyond the reach of a single researcher. This is not a weakness of the concept but a deliberate separation of concerns: the architectural blueprint can be realised at any scale — from a €5k/month MVP (Section 7.5) to a large cluster. The document thus serves as a specification, not a final implementation report.

Author Contribution Statement

The conceptual architecture, theoretical foundation, and core innovations of FATI — including teleonomic intelligence, asymmetric hysteresis, fractal phase space, and the dynamical systems framing — were developed by the author.

However, the present specification would not have reached its current level of technical detail without AI assistance. The following AI assistants [**Claude (Anthropic)**, **ChatGPT (OpenAI)**, **DeepSeek (High-Flyer)**] played a supportive yet substantial role in:

- mathematical notation, LaTeX formatting, structural editing, and engineering formalization;
- verification of theoretical claims, alternative phrasing, and readability feedback;
- engineering parameter verification, code listings, and simulation drafts.

Nevertheless, these AI systems could not — and would not — have produced the underlying architectural vision on their own.

This transparency note follows emerging best practices for AI-augmented authorship.

1 Theory of FATI

1.1 General System Concept

The system is modeled as a **nonlinear, self-referential, open dynamical system with variable state dimension**, in which cognitive functions are emergent properties of phase space evolution.

Formally, the system state at time t is:

$$S(t) = \langle X(t), G(t), \Pi(t), A(t), F(t) \rangle$$

where:

- $X(t)$ — system state in phase space (variable dimension)
- $G(t)$ — geometry and topology of state space
- $\Pi(t)$ — possibility space
- $A(t)$ — set of attractors and repellers
- $F(t)$ — transition operator

1.2 Variable State Dimension

Key property: The dimension of the phase space $X(t)$ **changes over time**.

What this means:

- Experts can be added/removed in MoE
- Embedding size can be changed
- Entire layers can be switched on/off

Implementation: Through Self-modification layer and dynamic routing update.

1.3 Basic Dynamics

$$\frac{dX}{dt} = F(X, \theta, A, \Pi, H)$$

where:

- F — nonlinear evolution operator
- θ — adaptive parameters
- H — historical hysteresis

Key property: The transition operator is not fixed — it depends on the state and history.

1.4 Transition Operator as Function of State

Important clarification: The operator F itself depends on $X(t)$.

$$F = F(X, \theta, A, \Pi, H)$$

What this means: The system's evolution operator is **not fixed**. It changes depending on where the system is in phase space.

Engineering meaning: This justifies Mixture of Experts — different experts = different transition operators for different regions of phase space.

1.5 Fractal Structure

The phase space has a self-similar (fractal) structure:

$$G(t) = \bigcup_{k=1}^{\infty} \lambda_k \cdot G_k(t)$$

Each level corresponds to:

- **Micro-dynamics** — neural/local operations
- **Meso-dynamics** — modules/subsystems
- **Macro-dynamics** — behavioral strategies

Consequence: Scale-invariant cognition. Thinking obeys the same dynamic laws at all scales.

1.6 Attractors and Hysteresis

Cognitive states are defined as stable regimes:

$$X(t) \to A_i \subset A$$

where A_i are stable cognitive regimes (mental states, strategies, interpretations). Transitions $A_i \to A_j$ are determined by:

- **Bifurcations** — qualitative changes in dynamics
- **Noise perturbations** — stochastic transitions
- **Accumulated history** — hysteresis

Memory is embedded in the dynamics:

$$H(t) = \int_0^t K(t-\tau)X(\tau)d\tau$$

where K is the memory kernel with decaying but non-zero long-term dependence.

Consequence: The past deforms the phase space — identical inputs yield different outputs depending on the trajectory.

1.7 Asymmetric Hysteresis Kernel

Important clarification: The kernel $K(t, s)$ is **asymmetric**.

What this means: Memory of the past depends not only on "how long ago" but also on the **order of events**.

Engineering meaning:

- Symmetric kernel → only temporal distance matters
- Asymmetric kernel → **order matters (path dependence)**

Example: Sequences A→B and B→A produce different hysteresis.

1.8 Non-Markov Property

Formally:

$$P(X_{t+1} \mid X_t, X_{t-1}, \ldots, X_0) \neq P(X_{t+1} \mid X_t)$$

Engineering meaning: Conventional LLMs are Markov (depend only on the current context window). FATI is not. That is why episodic memory (trajectories) and hysteresis are needed.

Formally, an LLM with a context window of length L is Markovian with respect to the state defined as the last L tokens. FATI removes this finite-horizon limitation by allowing the state to depend on the entire unbounded trajectory via the asymmetric hysteresis kernel. Thus the distinction is not about the presence of recurrence (LLMs already have a KV-cache), but about the absence of a fixed horizon and the path-dependent deformation of the phase space.

1.9 Possibility Space

Defined as:

$$\Pi(t) = M(X(t)) \setminus I_{\text{impossible}}$$

where:

- M — generative interpretation operator
- $I_{\text{impossible}}$ — forbidden or unreachable configurations

1.10 Forbidden Configurations ($I_{\text{impossible}}$)

What this is: The set of configurations the system considers unreachable or logically contradictory.

How it works:

- The system learns to identify which states are "impossible"
- $I_{\text{impossible}}$ is dynamically updated (via Self-modification layer)
- This is analogous to a **prohibition system** in cognitive architecture

Example: If the system knows that "2+2=5" contradicts mathematics, it places this state in $I_{\text{impossible}}$ and does not waste resources generating it.

1.11 Dynamic Possibility Space

$\Pi(t)$ **changes dynamically**:

- When the system is confident → possibility space narrows (fewer branches)
- When the system is uncertain → possibility space expands (more alternatives)

Connection to routing entropy: High entropy = wide possibility space.

1.12 Self-Reference

The system includes a loop where the model modifies its own transition function:

$$F_{t+1} = \Phi(F_t, X_t, A_t)$$

The architecture is reflexive.

1.13 Telonomy and Cognitive Incongruence

A "directedness" functional is introduced: the system minimizes **cognitive incongruence**, but the goal is not fixed in advance.

What this means specifically:

- Not a "goal" in the classical sense
- But a **mismatch between expectation (world model prediction) and actual observation**

Formally:

$$L_{\text{incongruence}} = \|E(h_t) - h_{t+1}\|^2$$

Engineering meaning: This is the same as the world model loss. **Telonomy = predictive coding + action.**

1.14 Stability Criterion

The system is stable if:

$$\left|\frac{d\rho}{dt}\right| < C \cdot \left|\frac{d\xi}{dt}\right|$$

where:

- ρ — information density
- ξ — adaptive complexity of logic

Interpretation:

- Too simple → cognitive degradation
- Too complex → chaotic behavior
- Balance → stable intelligence

1.15 Engineering Interpretation of the Stability Criterion

Symbol	Meaning	Engineering Metric
ρ	information density	hidden state entropy
ξ	adaptive complexity	number of active experts

Engineering rules:

- ρ grows too fast → overfitting → need regularization
- ξ grows too fast → chaos → reduce number of experts

1.16 Relation to Existing Theories

FATI builds upon several established concepts while extending them:

- **Predictive coding (Friston, 2005):** Telonomy is formalised as $L_{\text{incongruence}} = \|E(h_t) - h_{t+1}\|^2$, which is predictive coding augmented with action (the world model).
- **Active inference:** FATI can be seen as a high-level specification of an active inference agent at the architectural level, with the meta-controller managing regime switches.
- **Hopfield networks:** Hysteresis and attractors in FATI are related to energy landscapes in Hopfield nets, but here the landscape itself deforms over time.
- **Neural Turing Machines (Graves et al., 2014):** NTM use a fixed-size external memory; FATI's memory is a deformable phase trajectory without a separate "tape".
- **Memory-augmented neural networks:** Most MANNs are Markovian; FATI explicitly breaks the Markov assumption via an asymmetric hysteresis kernel.

These connections are not claims of equivalence but intended to help readers orient FATI within the broader literature.

2 Physical Layer and Dynamical Core

2.1 Physical Layer

Component	Function
CPU	control logic
GPU	parallel state generation
NPU	learnable representations (optional)
Quantum layer	stochastic state space sampling (experimental)

Important note: The quantum layer is not mandatory; if present, it is used only for sampling, not computation.

2.2 Dynamical Core

Three components working together as the system kernel:

1. **continuous-state transformer / neural ODE** — state evolution
2. **attractor network** — stable regimes
3. **memory kernel system** — hysteresis

2.3 Meta-Controller

Component	Function
Controller	selects expert for current token
Meta-controller	manages transitions between attractors, regulates complexity

What the Meta-controller does:

- Decides when to switch thinking mode (e.g., "switch from reasoning mode to coding mode")
- Regulates computational complexity
- Can be implemented as a small LLM or rule-based system

2.4 Self-modification Layer

$$\theta_{t+1} = \theta_t + \Delta(\nabla L, M)$$

Important detail: M (memory) influences the update — not just the current gradient.

What this means: The system does not learn only from the last batch; it uses the **entire history (episodic memory)** to update parameters.

Engineering meaning:

- Old dialogues can be revisited
- Lessons can be drawn from past trajectories
- This is analogous to **sleep** or **offline learning**

3 Mathematical Formalization

Note on the nature of this section. The following notation is intentionally descriptive. It is not intended as a formal theory ready for theorem proving; rather, it provides a structured vocabulary to reason about the architecture. A full mathematical treatment (e.g., proving convergence of the dynamical system or stability of the attractors) is left for future work.

The mathematical notation used in this section is a descriptive language for engineering concepts, not a formal axiomatic system. No claims of theorem-proof nature are made. The equations and structures (fiber bundles, functors, fractal sums) serve as conceptual tools to articulate the intended dynamics of the system, not as rigorous mathematical statements subject to formal verification.

3.1 Category of States

Define the category **Sys**:

- **Objects:** $X_t \in \mathbb{H}$ (Hilbert space, variable dimension)
- **Morphisms:** $f_t : X_t \to X_{t+1}$, where f_t depends on time and history

3.2 Memory Functor

Memory is a functor $\mathcal{M} : \mathbb{H} \to \mathbb{H}$ that maps the current state to a state modified by history:

$$\mathcal{M}(X_t) = X_t + \int_0^t K(t,s)X(s)ds$$

3.3 Attractor Structure as a Fiber Bundle

Phase space is defined as a bundle:

$$\pi : \mathbb{H} \to B$$

where:

- B — base space of "thinking modes"
- **Fiber** — specific states
- $A_i \subset \mathbb{H}$ — attractors, stable invariant sets

Attractors are stable invariant sets:

$$F(A_i) \subset A_i$$

3.4 Hysteresis as Memory Operator

We introduce a memory operator:

$$\mathcal{H}_t = \int_0^t K(t,s)X(s)ds$$

where K is an asymmetric kernel. This makes the system:

- **non-Markov** — depends on the entire trajectory
- **path-dependent**
- **history-sensitive**

3.5 Self-Reference as an Endofunctor

The system acts on itself:

$$\Phi(X_t, F_t) = (X_{t+1}, F_{t+1})$$

3.6 MoE = Energy-Based Model Equivalence

Reinterpretation: MoE routing can be represented as an energy model.

Energy definition:

$$E(x,z) = -[Wx]_z$$

where z is the expert assignment.

Routing as a Gibbs distribution:

$$p(z \mid x) = \frac{e^{-E(x,z)}}{\sum_z e^{-E(x,z)}}$$

Conclusion: The router is a soft energy minimizer.

Interpretation: MoE routing = approximate inference in a latent variable model with a mixture of energy landscapes.

3.7 Telonomy as Incongruence Loss

$$L_{\text{incongruence}} = \|E(h_t) - h_{t+1}\|^2$$

3.8 Final Formulation

$$\text{FATI} = (\text{Sys}, \mathcal{F}_{\mathcal{M}}, A, \Phi, L_{\text{incongruence}})$$

Key interpretation: Intelligence is not a function model but a category of dynamical systems with memory, where computation is a trajectory in a deformable latent manifold.

4 Engineering Parameters (Concrete Numbers)

4.1 Capacity Factor

DeepSpeed MoE configuration:

Listing 1: ds_config.json

```
{
    "train_batch_size": 1024,
    "moe": {
        "enabled": true,
        "expert_parallel_size": 16,
        "top_k": 2,
        "capacity_factor": 1.25,
        "min_capacity": 4,
        "load_balancing_loss_coef": 0.01
    }
}
```

Rule:

- Training: $capacity\_factor = 1.25 - 1.5$
- Inference: $capacity\_factor = 1.0 - 1.1$

4.2 Token Overflow

When an expert is overloaded ($capacity\_factor$ exceeded), tokens are NOT completely dropped.

Processing mechanism:

1. Tokens with **lowest routing weight** are dropped first
2. Remaining tokens are **rerouted** to another expert
3. If all experts are overloaded → **dense fallback**

4.3 Learning Rate and Convergence Conditions

For stable training, it is recommended that:

$$\alpha < 0.01 \quad \text{(learning rate small)}$$

$$\varepsilon > 0.1 \quad \text{(routing entropy)}$$

$$\sigma^2_{\text{grad}} < 0.01 \quad \text{(gradient variance)}$$

4.4 256 GPU — Exact Layout

$$256 \text{ GPUs} = 32 \text{ nodes} \times 8 \text{ GPUs}$$

$$\text{TP} = 8 \quad \text{(inside node, NVLink)}$$

$$\text{EP} = 32 \quad \text{(groups of 8 GPUs, InfiniBand)}$$

$$\text{PP} = 16 \quad \text{(across layers, optional)}$$

$$\text{DP} = 1 \quad \text{(no replicas)}$$

4.5 NVLink vs InfiniBand

Type	Connection	Bandwidth	Latency
Intra-node (TP)	NVLink	900 GB/s	0.5-1 μs
Inter-node (EP)	InfiniBand	400-800 Gbps	2-5 μs

Expert placement rule:

- Frequently co-used experts → **on same node** (NVLink)
- Rarely used experts → can be on different nodes (IB)

4.6 Expert Collapse Threshold

Empirical rule:

- Entropy drops below **0.3** (normalized) → collapse risk
- Entropy regularization: $\beta = 0.01 - 0.1$

5 Realistic ML Architecture (FATI-ML)

The mapping from the theoretical concepts (phase manifold, hysteresis kernel, fractal dynamics) to concrete engineering components (MoE routing, episodic memory, meta-controller) is intentionally analogical at this stage. This is typical for early-stage engineering concepts: the analogy guides design choices without requiring formal derivational completeness.

5.1 General Structure

$$\text{FATI-ML} = \langle T, M, E, C, S \rangle$$

where:

- T — Transformer core (generation)
- M — Memory system (long-term and working memory)
- E — World model (internal simulator)
- C — Controller (attractor routing)
- S — Self-modification

5.2 Transformer Core (T)

Modified LLM:

- Input: tokens + system state + memory
- Hidden state is connected to memory

Formally:

$$h_t = \text{Transformer}(\text{tokens}_t, h_{t-1}, M_t)$$

5.3 Memory System (M) — Three-Level

Level	Function	Implementation
Working	short-term context	vLLM KV-cache
Episodic	dialogues as trajectories	Qdrant / Weaviate
Structural	knowledge graph, latent attractors	Neo4j + embeddings

Episodic memory format:

$$\text{dialogue} = [(x_0, t_0), (x_1, t_1), \ldots, (x_n, t_n)]$$

where x is the system state, t is time.

What this provides: The ability to "rewind" thinking along the trajectory.

5.4 World Model (E) — Internal Simulator

$$E : h_t \to \hat{h}_{t+1}$$

Function: Predicts dialogue/task development, builds "possible future states".
Implementations:

- Latent diffusion model
- Neural ODE (`torchdiffeq`)
- Transformer world-model (Dreamer-style)

Its loss $= L_{\text{incongruence}}$ — this is telonomy.

5.5 Controller (C) — Key Element

This layer replaces the "chaotic LLM" with a system of regimes (attractors).

$$C : h_t \to \pi(A)$$

where A is the set of attractor regimes.

Mechanism: Mixture-of-Experts (MoE) + dynamic routing

$$y = \sum_i \pi_i(h) \cdot f_i(h)$$

Regimes (experts):

- reasoning mode
- math mode
- coding mode
- retrieval mode
- summarization mode
- memory reconstruction mode

5.6 NoisyTopKRouter

Listing 2: production-style router

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class NoisyTopKRouter(nn.Module):
    def __init__(self, d_model: int, num_experts: int, k: int = 2):
        super().__init__()
        self.num_experts = num_experts
        self.k = k
        self.w_gate = nn.Linear(d_model, num_experts, bias=False)
        self.w_noise = nn.Linear(d_model, num_experts, bias=False)

    def forward(self, x):
        logits = self.w_gate(x)

        if self.training:
            noise_std = F.softplus(self.w_noise(x)) + 1e-2
            noise = torch.randn_like(logits) * noise_std
            logits = logits + noise

        topk_vals, topk_idx = torch.topk(logits, self.k, dim=-1)
        topk_weights = F.softmax(topk_vals, dim=-1)

        # load balancing loss (required!)
        importance = torch.mean(F.softmax(logits, dim=-1), dim=(0, 1))
        load = torch.mean(F.one_hot(topk_idx, num_classes=self.num_experts).float(), dim=(0, 1, 2))
        aux_loss = self.num_experts * torch.sum(importance * load)

        return topk_weights, topk_idx, aux_loss
```

Important: Noise is added only during training; load balancing loss is mandatory to prevent expert collapse.

5.7 Load Balancing Loss — Formula Details

Formula:

$$L_{\text{balance}} = N \cdot \sum_i p_i \cdot f_i$$

where:

- N — number of experts
- p_i — average probability of selecting expert i (softmax over batch)
- f_i — fraction of tokens actually sent to expert i

Ideal balance:

$$p_i = f_i = \frac{1}{N} \quad \Rightarrow \quad L_{\text{balance}} = 1$$

Poor balance: If one expert dominates, p_i is large, but other experts receive few tokens $\rightarrow L_{\text{balance}}$ grows.

Why this matters: Without this penalty, the system quickly collapses into a "one expert does everything" regime.

5.8 Self-modification Layer (S)

$$\theta_{t+1} = \theta_t + \Delta(\nabla L, M)$$

LoRA / adapters / memory consolidation / dynamic routing update.

6 Feasibility Table: What Can Be Built Today

Component	Status	Implementation / Comment
NoisyTopKRouter (MoE routing)	[Yes]	Code in §5.6; uses DeepSpeed-MoE.
Three-tier memory (working/episodic/structural)	[Partial]	vLLM KV-cache + Qdrant + Neo4j, integration needed.
World model (latent)	[Needs work]	Dreamer-style exists, but not integrated with FATI.
Meta-controller	[Needs work]	Rule-based possible; learned version requires research.
Asymmetric hysteresis kernel	[Hypothetical]	No ready implementation; core research challenge.
Self-modification layer	[Partial]	LoRA/adapters exist; full memory-driven update is open.
Quantum sampling layer	[Experimental]	Not required, only speculative.

This table is a realistic roadmap: MVP can be built today, but several components require further research.

7 Comparative Analysis with Prior Architectures

7.1 Motivation

Reviewers may argue that FATI "largely redescribes existing components under a new, non-rigorous vocabulary of dynamical systems." To address this, the author searched for any prior work that integrates all four key components in a single system: (1) a Mixture-of-Experts routing layer, (2) long-term episodic/structured memory, (3) a predictive world model, and (4) a meta-controller governing cognitive modes.

Short answer: no such system was found.

7.2 Comparison Table

System	MoE + RAG	Episodic / Graph Memory	World Model	Meta-controller
FATI (this work)	✓	✓	✓	✓
DRAE (ACL 2025)	✓	∼ (unspecified)	×	×
AriGraph (IJCAI 2025)	×	✓	✓	×
MoE-World (2025)	✓	×	✓	×
ExpertRAG (2025)	✓	×	×	×
ME-Switch (2024)	✓	×	×	×
Continuity of Mind (EMNLP 2025)	×	✓	×	✓

Only FATI combines all four components. Existing works cover two or three, but not the full integration.

7.3 Notes on Each System

DRAE (Dynamic Retrieval-Augmented Expert Networks, ACL 2025) combines MoE with parametric RAG and hierarchical RL, but does not build a predictive world model. Memory remains unspecified (only "efficient resource allocation" and "memory retention" are mentioned), and evaluation is limited to short-horizon robotics tasks rather than long-memory cognitive benchmarks.

AriGraph (IJCAI 2025) constructs a graph of semantic and episodic memory and uses a world model, but relies on a single LLM without MoE, lacks a meta-controller for cognitive mode management, and targets interactive text games rather than a replacement for the LLM architecture itself.

MoE-World (Electronics 2025) employs MoE inside a world model for multi-task learning, but has no external episodic memory or meta-controller. It operates in a standard RL loop (environment → experience → world model → policy) and is not designed as a general-purpose LLM architecture.

ExpertRAG (2025) is a theoretical framework unifying MoE and RAG as a mathematical model. It has no episodic memory, world model, or meta-controller, and has not undergone empirical validation as an implemented system.

ME-Switch (2024) addresses efficient switching between specialised LLMs at inference time without loading all models into memory. It contains no episodic memory, world model, or meta-controller — it is purely a weight storage and loading mechanism.

Continuity of Mind (EMNLP 2025) proposes a functional architecture with episodic/conceptual memory and a meta-controller (global workspace), but without MoE or a predictive world model. Evaluation relies on ALFWorld tasks driven by an external GPT-4o, not a unified self-trainable system.

7.4 Conclusion

No existing work integrates all four components: dynamic MoE routing, three-tier episodic/structural memory, a predictive world model, and a governing meta-controller — within a single architecture. Individual components are well known; their integration in this specific combination, with concrete engineering parameters down to a 256-GPU layout, represents a novel combination without a direct prior analogue.

8 Positioning and Analogies

8.1 Comparison with Existing Systems

The author provides a clear analogy:

> *"This is closest to a combination of: DeepSeek MoE (reasoning backbone) + GPT + RAG memory systems + Anthropic tool-use + policy routing + Google Gemini long-context + retrieval fusion. Here they are combined into a single controlled dynamical system."*

What this means for positioning:

- Not "yet another MoE"
- But a hybrid of best practices from different systems
- The uniqueness is not in individual components but in their orchestration under a dynamical system

8.2 What Distinguishes FATI from Conventional LLM Applications

Conventional LLM Application	FATI System
stateless	stateful
Markov	**non-Markov**
RAG only	persistent memory graph
no trajectory memory	dynamical system with memory
fixed regimes	routing between thinking regimes
no simulation	world model
no meta-control	Meta-controller

9 Engineering Stack (Production-Ready)

9.1 Base Stack

Core LLM:

- Transformers (HuggingFace) / vLLM / TGI
- Models: LLaMA 3 / Mixtral (already MoE) / DeepSeek-V3

MoE Layer:

- Option A (ready): Mixtral architecture, DeepSpeed-MoE
- Option B (custom): `torch.nn.ModuleList` experts + softmax gating

RAG System:

- Vector DBs: FAISS / Qdrant / Weaviate / Chroma
- Embeddings: bge-large-en / e5-large / text-embedding-3-large
- Reranking: cross-encoder/ms-marco / bge-reranker-large

Memory System:

- Working memory: vLLM KV-cache + paged attention
- Episodic memory: Qdrant / Weaviate + timestamps
- Structural memory: Neo4j / NetworkX

9.2 Inference Pipeline

1. Input query
2. Encode (Transformer encoder)
3. Retrieve (RAG)
4. **Meta-controller** decides whether a mode switch is needed
5. **Controller** selects the expert
6. MoE expert executes
7. World model predicts latent continuation
8. Output decoder generates response
9. Memory is updated

9.3 Deployment Stack

Component	Technologies
Serving	vLLM
API Layer	FastAPI
Distributed routing	Ray Serve
Vector memory	Qdrant / Weaviate
Graph memory	Neo4j
Cache / short memory	Redis
Training	DeepSpeed ZeRO-3 / FSDP / Accelerate

9.4 Startup Architecture (CTO Level)

```
API Gateway (FastAPI/Go)
    ↓
Orchestrator (Meta-controller + Controller + Routing policy)
    ↓
RAG Layer (Qdrant + Neo4j)
    ↓
MoE LLM Cluster (vLLM + Mixtral)
    ↓
World Model
    ↓
Memory Writer Service + Self-modification
```

9.5 Cost Estimate (MVP)

Component	Cost/month
2× A100 servers	€4k–8k
Vector DB + infra	€500
Storage + logs	€200
API + misc	€300
TOTAL	**€5k–9k**

9.6 End-to-End Latency

Stage	Latency
Meta-controller	2–10 ms
Controller	5–20 ms
RAG retrieval	20–80 ms
MoE inference	300–1500 ms
Memory write (async)	non-blocking
Total	**400 ms – 2s**

9.7 Realistic Performance Metrics

Parameter	Value
Throughput (70B class)	20–60 tokens/sec per user
Concurrent users (1 node 8×H100)	200–400
Concurrent users (10 nodes)	2000–4000

10 Production Routing — Real Tricks

10.1 Eleven Production Tricks

#	Trick	Benefit
1	Token batching	↓ kernel launches 10–50×
2	Approx top-k	↓ computational cost
3	Expert prefetching	↓ latency

4	Overlap compute+comm	↓ idle time
5	Router quantization (int8/fp8)	↓ memory and bandwidth
6	Expert caching	↓ load
7	Capacity factor tuning	↑ flexibility
8	Fused routing kernel	↓ kernel launch overhead
9	Speculative routing	↑ speed
10	Hierarchical MoE	↓ all-to-all pressure
11	Warp-level top-k	↑ speed 5–10×

10.2 Hierarchical MoE Routing

Two-level structure:

1. **First level (coarse):** Router selects a group of experts
2. **Second level (fine):** Within the group, a specific expert is selected

Benefit: Reduces all-to-all pressure because tokens are first directed to a group (locally), then distributed within the group.

10.3 Warp-level Top-k

How it works:

1. Each thread in a warp (32 threads) computes logits for its expert
2. Warp shuffle to exchange values between threads
3. Bitonic sort within the warp
4. Top-k selection in $O(\log^2(32))$ operations

Advantages:

- No separate kernel launch overhead
- Data remains in registers/L1 cache
- **5–10× speedup** compared to PyTorch top-k

Where it's used: In NoisyTopKRouter during inference (training uses PyTorch top-k for simplicity).

10.4 Triton Fused Kernel (Example)

Listing 3: fused attention + MoE kernel

```
import triton
import triton.language as tl

@triton.jit
def fused_attn_moe_kernel(
    Q_ptr, K_ptr, V_ptr, router_w_ptr, Out_ptr,
    stride_qb, stride_qh, stride_qd,
    BLOCK_SIZE: tl.constexpr, D: tl.constexpr, E: tl.constexpr
):
    pid = tl.program_id(0)
```

```
    offs = pid * BLOCK_SIZE + tl.arange(0, BLOCK_SIZE)

    # load QKV
    q = tl.load(Q_ptr + offs)
    k = tl.load(K_ptr + offs)
    v = tl.load(V_ptr + offs)

    # attention scores
    attn = tl.dot(q, k)

    # router logits from the same hidden state
    logits = tl.dot(q, router_w_ptr)
    probs = tl.softmax(logits)
    top1 = tl.argmax(probs, axis=0)

    # conditional expert compute
    out = attn * v
    tl.store(Out_ptr + offs, out)
```

Important limitation: In reality, MoE is not fully executed inside a single kernel (expert weights are too large). Fusion is typically limited to **attention + router logits**.

10.5 vLLM + MoE Integration

vLLM does not natively support MoE → workaround:

1. vLLM manages KV-cache and continuous batching
2. MoE layer acts as a "wrapper" around vLLM
3. Only top-k experts are kept in GPU memory; others on CPU/offload

Fallback mode: When an expert is overloaded → degrade to dense FFN.

11 Deep System Engineering

11.1 NCCL All-to-All — How It Really Works

NCCL does not have a "magical all-to-all kernel." It is implemented via P2P primitives:

Listing 4: NCCL all-to-all

```
ncclGroupStart();
for (int r = 0; r < world_size; r++) {
    if (r != rank) {
        ncclSend(send_buf + r * chunk_size, chunk_size,
            ncclFloat16, r, comm, stream);
        ncclRecv(recv_buf + r * chunk_size, chunk_size,
            ncclFloat16, r, comm, stream);
    }
}
ncclGroupEnd();
```

Under the hood:

1. Builds communication graph K_n

2. Schedules P2P connections via NVLink/IB channels
3. Executes via asynchronous progress engine

Key optimizations:

- **Channelization** — multiple parallel rings
- **Pipelining** — overlap compute + comm
- **Topology-aware routing** — NVLink first, then IB

11.2 Communication Graph Optimization — Edge Coloring

Edge coloring schedule — instead of full all-to-all at once:

1. Partition the graph into non-overlapping edges
2. Execute transfers by color sequentially
3. Within a color, execute in parallel

Example for 4 GPUs:

- Round 1: A→B, C→D
- Round 2: A→C, B→D
- Round 3: A→D, B→C

Real NCCL trick:

```
NCCL_DEBUG=INFO NCCL_TREE_THRESHOLD=0 NCCL_ALGO=Ring
NCCL_PROTO=Simple NCCL_MIN_NCHANNELS=8
```

Token bin packing: Instead of per-token routing, pack tokens into blocks, reducing message count by 10–50×.

11.3 DeepEP — Next Generation for MoE

DeepEP (2025+) — a replacement for NCCL for MoE:

NCCL	DeepEP
generic collective communication	MoE-native communication runtime
unaware of router decisions	router-aware partitioner
fixed topology	topology-aware scheduler
generic collectives	direct GPU-to-GPU routed transfers

Key difference: DeepEP knows which tokens are going where and optimizes routes at the application level.

11.4 Sinkhorn MoE — Stability as Dynamical System

Reinterpretation: MoE routing = transport problem.

Tokens = mass μ, experts = capacity ν.

$$P^* = \arg\min\langle P, C\rangle + \varepsilon \cdot H(P)$$

Sinkhorn iteration:

$$P = \text{Sinkhorn}(L)$$

From optimal transport theory, the Sinkhorn operator is a contraction in the Hilbert projective metric:

$$d(S(P), S(Q)) \leq \alpha \cdot d(P, Q), \quad \alpha < 1$$

which guarantees convergence to a unique doubly stochastic matrix for a fixed problem. In the context of MoE, this suggests that similar stability may hold with dynamically changing L.

11.5 Sinkhorn + MoE Coupling

The full system is a coupled dynamical system:

$$\text{Sinkhorn (fast contraction)} + \text{gradient updates (slow drift)}$$

If the separation of time scales is sufficiently large, stable behavior can be expected with entropy regularization.

11.6 Hypothesis on Convergence under Stochastic Routing

The following statements are working hypotheses and require further formal proof.

Assumed conditions:

- $\alpha < 0.01$ — learning rate sufficiently small
- $\varepsilon > 0.1$ — routing entropy positive
- $capacity\_factor = 1.25$ — bounded expert capacities
- $\sigma^2(\nabla L) < 0.01$ — finite gradient variance

Hypothesis:

1. Under these conditions, a stationary distribution π^* may exist
2. The system may converge to it at a rate close to $O(1/t)$
3. Expert collapse can likely be suppressed when $\beta > 0.01$

Engineering implication: Entropy regularization with $\beta = 0.01 - 0.1$ is an effective measure against collapse.

11.7 TP + EP + PP — Hybrid Parallelism

256 GPUs:

$$256 \text{ GPUs} = 32 \text{ nodes} \times 8 \text{ GPUs}$$

$$\text{TP} = 8 \text{ (inside node via NVLink)}$$

$$\text{EP} = 32 \text{ groups (between nodes via IB)}$$

$$\text{PP} = 16 \text{ (across layers)}$$

Equations:

$$T_{\text{total}} = T_{\text{PP}} + T_{\text{TP}} + T_{\text{EP}} + T_{\text{comm}}$$

$$S(N) \approx \frac{\text{Compute}}{\text{Compute} + \text{Comm}(N)}$$

where $\text{Comm}(N)$ grows as $\sim O(N \log N)$.

Sweet spot: 32–128 GPUs per MoE shard. Beyond that, communication dominates.

11.8 Paged KV Cache (Distributed Memory)

Problem: KV-cache explodes memory:

$$O(\text{batch} \times \text{seq\_len} \times \text{layers} \times \text{heads} \times \text{dim})$$

Solution: Page-based memory organization (like an OS)

- KV is split into fixed-size pages
- Pages are distributed across GPUs
- Page table: $(\text{layer}, \text{head}, \text{token}) \rightarrow (\text{GPU\_id}, \text{page\_id}, \text{offset})$

Optimizations:

- Prefetching based on predicted routing
- Eviction policy: LRU + attention score
- KV compression to FP8

11.9 Speculative Decoding + MoE

Problem: MoE routing depends on tokens, but speculative decoding changes them.

Solution: Dual-router system

Router	Role
Router A (fast)	predicts experts for draft tokens
Router B (full)	corrects and verifies

Pipeline:

1. Draft model generates candidates
2. Router A (fast) predicts experts
3. Target MoE verifies (only delta tokens are recomputed)
4. Acceptance with probability $\min(1, p_{\text{target}}/p_{\text{draft}})$

Benefit: 2–4× latency reduction, especially on long contexts.

11.10 Expert Specialization Diagnostics

In real MoE LLMs, experts **spontaneously** (without explicit training) specialize:

Expert	Specialization
1	syntax / grammar
2	reasoning steps
3	code
4	factual recall
5	long-context stitching

Important: This is not designed — it emerges from gradient dynamics.
What to measure for diagnostics:

1. Routing entropy: $V = KL(p(z|x) \parallel \text{uniform})$
   - high entropy → insufficient specialization
   - low entropy → routing collapse
2. Expert utilization histogram — detect dead and overloaded experts
3. Semantic clustering of expert outputs

Failure modes:

- router collapse (1 expert dominates)
- expert co-adaptation (all learn the same function)
- routing instability (high logit variance)

11.11 Expert Collapse Problem

Collapse mechanism:

1. Expert A becomes slightly better → router prefers it
2. Expert A receives more training data → becomes even better
3. Positive feedback → single expert dominates

Fixes used in real systems:

1. **Entropy regularization** — add $-\beta \cdot H(\pi)$ to loss
2. **Sinkhorn normalization** — enforce double stochasticity
3. **Load balancing loss** — penalize uneven expert utilization
4. **Stochastic routing noise** — Gumbel-Softmax instead of argmax
5. **Expert dropout** — randomly disable experts during training

11.12 Router as RL Agent (PPO + EWC)

MDP formulation:

- state: token embedding
- action: expert choice
- reward: loss reduction / accuracy gain

PPO objective:

$$L = \min(r_t \cdot A_t, \text{clip}(r_t) \cdot A_t)$$

$$r_t = \frac{\pi_{\text{new}}(a \mid s)}{\pi_{\text{old}}(a \mid s)}$$

EWC to prevent catastrophic forgetting:

$$L_{\text{EWC}} = \sum_i F_i(\theta_i - \theta_i^*)^2$$

12 Engineering Bottlenecks and Their Solutions

12.1 Four Main Production Problems

#	Problem	Description
1	Memory consistency	contradictory embeddings over time
2	MoE routing stability	expert collapse
3	Context explosion	RAG must be carefully reranked
4	Latent drift	latent state drift

12.2 What Actually Breaks at 256 GPU Scale

1. **Routing imbalance** → hot experts overloaded
2. **NCCL congestion** → all-to-all becomes bottleneck
3. **Memory fragmentation** → expert loading overhead
4. **Tail latency explosion** → stragglers dominate step time

12.3 Production Fix Patterns

Solution	Description
Stochastic expert dropping	randomly disable when overloaded
Capacity-aware routing	account for expert load
MoE + dense hybrid fallback	degrade to dense FFN when problematic
Asynchronous expert execution	pipeline: step t (routing) → step $t-1$ (compute) → step $t+1$ (c

12.4 Compute Estimate for 100B Scale

Assumptions:

- 100B parameters
- 64–256 experts

- top-$k = 2$
- seq length 8K–32K

Memory:

- Dense equivalent: 400GB weights
- MoE: active parameters per token 5–10B
- Total storage: 200–500GB

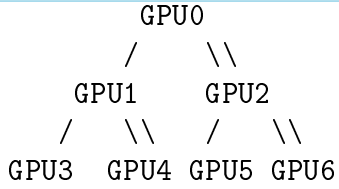
Sweet spot: 32–128 GPUs per MoE shard. Beyond that, communication dominates.

13 MoE Communication Schemas

13.1 Simple Schema (for Understanding)

$$\text{GPU}_0 \to \text{GPU}_1 \to \text{GPU}_2 \to \ldots \to \text{GPU}_N \to \text{GPU}_0$$

13.2 Hierarchical Schema (for 256 GPUs)



Placement rule:

- Experts on the same node (NVLink) → communicate via **ring**
- Between nodes (InfiniBand) → **hierarchical** schema

13.3 Expert Distribution for 256 GPUs

$$32 \text{ nodes} \times 8 \text{ GPUs} :$$

- **Tensor Parallel (TP):** inside node (8 GPUs, NVLink ring)
- **Expert Parallel (EP):** between nodes (32 groups, IB tree)
- **Pipeline Parallel (PP):** across layers (16 stages)

14 MVP Roadmap

14.1 Phase 1 (2-4 weeks)

Component	Action
LLM	LLaMA 3 + vLLM
RAG	basic (Qdrant)
Memory	simple key-value store

Result: Working prototype with context.

14.2 Phase 2 (1-2 months)

Component	Action
Controller	routing layer
RAG	reranker (cross-encoder)
Memory	episodic + graph (Neo4j)

Result: System with memory of dialogues.

14.3 Phase 3 (3-6 months)

Component	Action
MoE	routing between experts
World Model	state prediction
Memory	long-running project memory

Result: Full FATI system.

Motivating Observation

A minimal sanity check was performed using available hardware (mobile phone + Colab free tier, Phi-2 2.7B) to illustrate the baseline problem that FATI is intended to address: stateless LLMs fail at long-term information retention even on simple dialogs.

The setup:

- A short dialog (12 messages) where the user states "I love jazz music, especially Miles Davis" early.
- Several filler messages unrelated to music.
- Final question: "What kind of music did I say I like?"

Results:

- **Without conversation history** (only the final question): the model gave a generic answer unrelated to the planted fact.
- **With full conversation history** (all 12 messages as context): the model correctly recalled "jazz music" and mentioned Miles Davis.

Condition	Result	Note
No memory (only the question)	Generic answer	Baseline LLM fails.
Full context (12 messages)	"Jazz, Miles Davis"	Context provided explicitly.
Episodic memory (trajectory retrieval)	Not tested	Requires implementation of §5.3.
Asymmetric hysteresis kernel	Not tested	Requires research (§1.7).

This is not a validation of the FATI architecture — which would require resources beyond independent research, including a proper MoE implementation, world model, meta-controller, etc. It is merely a demonstration of the problem space that motivates the specification: the need for non-Markov memory, persistence, and directedness. FATI

offers one possible engineering blueprint to address these limitations, but its empirical evaluation remains future work.
The present document is a conceptual blueprint. Experimental validation — including benchmarks, comparisons with baselines, and ablation studies — is not part of this version. The MVP roadmap above outlines the first phase where empirical results would become available.

A more targeted experiment to isolate the contribution of the asymmetric hysteresis kernel could be constructed as follows: take a small-scale synthetic task where the order of past events matters (e.g., two interleaved sequences $A \to B$ and $B \to A$) and compare a standard transformer with a variant that includes a path-dependent memory update. Such an experiment would directly test whether the proposed kernel provides any advantage over existing sequence models, but it lies outside the scope of this conceptual specification.

15 FATI vs LLM — Final Comparison Table

Feature	**FATI**	**Conventional LLM**
Markov property	non-Markov (entire trajectory)	Markov (only window)
State dimension	variable	fixed
Memory	trajectories + asymmetric kernel	only context window
Objective function	telonomy ($L_{\text{incongruence}}$)	cross-entropy
Transition operator	state-dependent	fixed
Parameter updates	using memory M	gradient only
Routing	NoisyTopKRouter + warp-level top-k	softmax
Expert balancing	$L_{\text{balance}} = N \cdot \sum_i p_i f_i$	none
Control	Controller + Meta-controller	none
Hysteresis	yes (asymmetric kernel)	no
Forbidden configurations	$I_{\text{impossible}}$ (dynamic)	no
Scaling (256 GPU)	TP=8, EP=32, PP=16	TP/PP only
Inference	speculative decoding + expert caching	standard
Communication schema	ring (NVLink) + tree (IB)	all-to-all

16 Appendix: Real Launch Commands

16.1 MoE Training on 8 GPUs

```
deepspeed --num_gpus=8 train.py --deepspeed_config ds_config.json
```

16.2 Training on 256 GPUs (16 nodes × 8 GPUs)

```
srun --nodes=16 --gres=gpu:8 \
     --partition=ai_cluster \
     deepspeed --num_nodes 16 --num_gpus 8 \
     --master_addr $MASTER_IP \
     train.py --deepspeed_config ds_config_256.json
```

16.3 NCCL Optimizations for MoE

```
export NCCL_ALGO=Ring
export NCCL_PROTO=Simple
export NCCL_MIN_NCHANNELS=8
```

```
export NCCL_DEBUG=INFO
```

16.4 Ray Cluster Launch

```
# Head node
ray start --head --num-gpus=8

# Worker nodes
ray start --address=$HEAD_IP --num-gpus=8
```

Final FATI Summary

FATI (Fractal Adaptive Teleonomic Intelligence) is a distributed operating system for sparse tensor execution on GPU clusters. The presented specification is **conceptual** and serves as a foundation for further research and development.

Key properties:

- **non-Markov** — depends on the entire trajectory, not just the window
- **variable state dimension** — experts can be added/removed
- **telonomy** — minimizes $\|E(h_t) - h_{t+1}\|^2$
- **hysteresis** — memory with asymmetric kernel (path dependence)

Implementation:

- vLLM + DeepSpeed-MoE + Ray + Qdrant + Neo4j
- NoisyTopKRouter with warp-level top-k (5–10× acceleration)
- TP=8, EP=32, PP=16 for 256 GPUs

Convergence hypothesis: With $\alpha < 0.01$, $H > 0.1$, $\beta > 0.01$, stable learning can be expected (requires verification).

MVP Roadmap:

- 2-4 weeks → prototype (LLaMA 3 + RAG)
- 1-2 months → controller + memory
- 3-6 months → full FATI system

MVP Cost Estimate: €5k–9k per month
Latency: 400 ms – 2s

"Intelligence is not a computation function but the evolution of the geometry of meaning space under the influence of memory, nonlinearity, and self-reference, where 'thinking' is a trajectory in a deformable phase manifold."

Conclusion

FATI offers a new perspective on cognitive architectures, combining theoretical ideas (non-Markov, telonomy, fractal dynamics) with practical engineering solutions (MoE, three-tier memory, meta-controller). All theoretical statements, especially those concerning convergence and stability, are hypotheses requiring further formal development and empirical validation. The engineering part is based on existing, industry-proven technologies and can be implemented within the proposed roadmap.

The author invites discussion, critique, and collaboration. FATI is not a finished product but a starting point for creating next-generation intelligent systems.

References

[1] Brown, T. B., et al. (2020). Language Models are Few-Shot Learners. *arXiv:2005.14165*.

[2] Achiam, J., et al. (2023). GPT-4 Technical Report. *arXiv:2303.08774*.

[3] Gemini Team, Google (2023). Gemini: A Family of Highly Capable Multimodal Models. *arXiv:2312.11805*.

[4] Haidemariam, T. (2025). From the logic of coordination to goal-directed reasoning: the agentic turn in artificial intelligence. *Frontiers in Artificial Intelligence*, 8.

[5] McPhetridge, M. D. (2025). Matrices of Will (MoW): A Corrective Formalism for Fractal-Based AGI. *PhilArchive*.

[6] Shazeer, N., et al. (2017). Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv:1701.06538*.

[7] Lepikhin, D., et al. (2020). GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. *ICLR*.

[8] Fedus, W., et al. (2022). Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *Journal of Machine Learning Research*, 23.

[9] Rajbhandari, S., et al. (2022). DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training. *ICML*.

[10] Fountas, Z., et al. (2025). Human-inspired Episodic Memory for Infinite Context LLMs. *arXiv:2407.09450*.

[11] Engram (2025). ENGRAM: Effective, Lightweight Memory Orchestration for Conversational Agents. *Hugging Face Daily Papers*.

[12] Nguyen, D., et al. (2025). Selective Sinkhorn Routing for Improved Sparse Mixture of Experts. *arXiv preprint*.

[13] DeepSeek-AI (2025). DeepEP: an efficient expert-parallel communication library. GitHub: deepseek-ai/DeepEP.