A Systems and Network-Theoretic Framework for Large Language Models and Compound AI Systems

An Engineering Approach to Design, Measurement, and Communication

Kevin R. Haylett
Manchester, UK

Selected Communications

July 2026

Preface

This document is written for engineers, designers, and researchers who work with large language models (LLMs) and the compound systems built from them. It is also written for anyone who has encountered the language currently used to describe these systems—"agents think", "models reason", "systems have goals"—and sensed that something is wrong.

The thesis of this document is straightforward:

> Large language models are nonlinear dynamical systems. Compound systems built from them are networks of nonlinear dynamical systems. The language we use to describe them should reflect this engineering reality, not obscure it.

This is not a philosophical position. It is a pragmatic engineering claim. When we use imprecise, anthropomorphic language to describe these systems, we:

- obscure the underlying mathematics;
- hinder debugging and diagnosis;
- confuse regulatory and safety discussions; and
- distance the field from rigorous engineering disciplines.

This document offers an alternative: a precise, portable, engineering-based vocabulary drawn from systems theory, control theory, and network theory. It is offered as a working foundation, not a finished theory. Its value, if it has any, is in supplying a disciplined language for a class of engineering practice that is currently conducted, in large part, informally.

The document is structured as a textbook-style introduction. It begins with the core concepts, builds the formalism layer by layer, applies it to real engineering problems, and closes with open questions and a research programme. It is designed to be read sequentially.

Contents

Part I

Foundations

Chapter 1

The Problem of Language

1.1 What Are We Actually Building?

When an engineer builds a bridge, they do not say the bridge "wants" to stand, "decides" to distribute load, or "understands" the forces acting upon it. They describe the bridge in the language of structural mechanics: forces, stresses, moments, deflections. The language is precise, measurable, and shared across the discipline.

When an engineer builds a system around a large language model, they often describe it differently. They say:

- "The agent thinks about the problem"
- "The model reasons through the steps"
- "The system reflects on its output"
- "The agent pursues its goal"

This is not engineering language. It is anthropomorphic shorthand—language borrowed from human cognition to describe what is, in fact, a mathematical operation.

The shorthand creates a very specific framework that directs our thinking. Once we say "the agent thinks", we begin to treat the system as though it has thoughts. Once we say "the model reasons", we begin to expect reason-like behaviour. Once we say "the system reflects", we begin to attribute reflective capacity where none exists.

This may be described as semantic drift: the gradual movement from precise description to imprecise metaphor, from what the system does to what we imagine it does.

1.2 The Gap Between Language and Reality

Consider what a large language model actually is:

- a matrix of numerical weights, typically billions or trillions of parameters;
- a fixed transformation that maps an input token sequence to an output probability distribution;

- a function F that is static—it does not change during operation; and
- a system with no internal state beyond the input it receives.

When we call this system an "agent" that "thinks" and "decides", we are using language that implies:

- persistence of self over time (there is no self);
- internal deliberation (there is no deliberation);
- intentionality (there is no intention); and
- agency (there is no agency).

The system does none of these things. It transforms inputs to outputs according to a fixed transfer function. That is all.

The problem is not that the shorthand is used. Shorthand is useful, however the problem is that the shorthand has become the operational frame for many practitioners, regulators, and members of the public. The metaphor has become a collective 'truth' that may be misleading and ineffective for practical daily purposes and engineering design.

1.3 Why This Matters for Engineering

This may seem an abstract concern. However it has concrete engineering consequences:

Problem	**Consequence**
We say the system "hallucinates"	We attribute a cognitive failure to a statistical process. We look for cognitive fixes when we should look for statistical fixes.
We say the system "forgets"	We attribute memory failure to a context-window limitation. We design memory mechanisms when we should design context management.
We say the system "reflects"	We attribute recursive self-evaluation to what is often just another call to the same function. We build "reflection" loops when we should build measurement and verification.
We say the system "pursues goals"	We attribute intentionality to what is actually a deterministic control loop. We design "goal-setting" when we should design reference signals and error functions.

It seems to me that this engineering cost of this drift is real as it leads to:

- misdiagnosed failures;
- inefficient debugging;
- over-engineered "solutions" to non-existent problems; and
- under-engineered solutions to real problems.

1.4 The value of LLMs and AI systems

Importantly, this document is not a critique of LLMs or rejection of the technology. Nor is t an argument that these systems are “not intelligent” or “not useful”. This exposition simply presents an argument that the language we use to describe these systems should match the engineering reality. The systems are measurably useful and powerful. They are transforming what is possible in computation. However, they are not agents, minds, or thinkers. They are perhaps best considered as nonlinear dynamical systems. This perspective suggests such systems should be engineered as such.

Chapter 2

Core Concepts

2.1 What Is a System?

Before we can describe LLMs and compound AI systems in engineering terms, we must define what we mean by a system.

> **Definition 2.1 — System**
> A system Sys is a triple
> $$\mathsf{Sys} = (\mathcal{I}, \mathcal{O}, \Phi),$$
> where:
> - $\mathcal{I}$ is the input space;
> - $\mathcal{O}$ is the output space; and
> - $\Phi : \mathcal{I} \to \mathcal{O}$ is a transformation.

This definition says nothing about what happens inside the system. The input enters, the output exits, and the transformation occurs. Everything inside is, at this level of description, a black box.

This is the standard discipline of engineering systems analysis: define the ports first, and only then open the box.

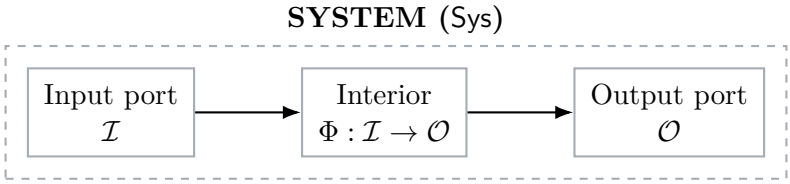

Figure 2.1: The System Boundary

Figure Description: A rectangular box represents a system with a defined boundary. The input port $\mathcal{I}$ feeds the interior transformation $\Phi : \mathcal{I} \to \mathcal{O}$, which feeds the output port $\mathcal{O}$. At the system level, only the relationship between inputs and outputs is required; the interior may be treated as a black box.

2.2 What Is a Filter?

In signal processing and systems theory, a filter is a system that transforms an input signal into an output signal according to a specific transfer function.

> **Definition 2.2 — Filter**
> A filter is a system whose transformation Φ is:
> - **static**: it does not change during operation;
> - **deterministic**: given the same input, it produces the same output (stochasticity, where present, is external to the filter); and
> - **characterised by a transfer function**: the mathematical relationship between input and output.

A large language model, at its core, is precisely this:

- **Input:** a sequence of tokens (the context window);
- **Output:** a probability distribution over the vocabulary (or, after decoding, a token sequence); and
- **Transfer function:** the fixed weight matrix, determined by training and fixed thereafter.

The LLM is a static, nonlinear, high-dimensional filter. It does not learn during operation. It does not adapt. It does not change. It is a fixed transformation that maps inputs to outputs according to a fixed transfer function.

> **Figure Description:** The input stream is the accumulated token sequence $S(t) = [S_0, p_1, r_1, p_2, r_2, \ldots, p_t]$. It flows through a fixed nonlinear transform F, producing logits $y(t) = F(S(t))$. Decoding then maps the output distribution to a response r_t. The diagram emphasises that the trained transform is fixed during operation and has no internal memory beyond the supplied input.

2.3 The State Vector

The LLM filter operates on a state vector—the accumulated history of the conversation.

> **Definition 2.3 — State Vector**
> The state vector $S(t)$ at step t is the full content of the input port at that step:
>
> $$S(t) = S_0 \oplus p_1 \oplus r_1 \oplus p_2 \oplus r_2 \oplus \cdots \oplus p_t,$$
>
> where:
> - S_0 is the system prompt (the initial boundary condition);
> - p_i is the i-th user prompt;
> - r_i is the i-th LLM response; and
> - $\oplus$ denotes ordered concatenation.

The state vector grows monotonically as the conversation proceeds. Each new prompt and response is appended to the existing state.

Key observation: The LLM filter has no memory of its own. All "memory" is represented explicitly in the state vector. The filter does not remember; the state vector contains the history. This is a critical distinction that the anthropomorphic language obscures.

2.4 Static and Dynamic Capacity

Every filter has capacity constraints. For an LLM, there are two distinct types of capacity, and they must not be conflated.

> **Definition 2.4 — Static Capacity**
> Static capacity is the set of properties fixed at construction time:
> - parameter count (number of weights);
> - architecture (layers, attention heads, hidden dimensions);
> - training corpus (the data used to train the model); and
> - training process (algorithm, duration, hyperparameters).
>
> Static capacity does not change during deployment. It is what a specification sheet reports.

> **Definition 2.5 — Dynamic Capacity**
> Dynamic capacity is the residual capacity available at a given call:
>
> $$C_{\text{dynamic}}(t) = C_{\max} - |S(t)|,$$
>
> where:
> - $C_{\max}$ is the maximum context length (the maximum number of tokens the filter can process in one forward pass); and
> - $|S(t)|$ is the current state-vector length (the number of tokens currently in the context).
>
> Dynamic capacity is consumed monotonically as the conversation proceeds.

Figure Description: Static capacity consists of construction-time properties such as parameter count, architecture, training corpus, and maximum context length. Dynamic capacity is the residual context capacity $C_{\max} - |S(t)|$, which declines as the state vector grows.

2.5 Static vs. Dynamic: A Critical Distinction

The distinction between static and dynamic capacity is one of the most basic and most frequently blurred in informal discussion of these systems.

Claim	What It Actually Means
"The model is capable of reasoning"	Static capacity claim: the weights have learned patterns that sometimes appear reasoning-like.
"The model is losing track of the conversation"	Dynamic capacity claim: the context window is approaching saturation.

Claim	**What It Actually Means**
"The model is intelligent"	Static capacity claim: the weights encode useful patterns.
"The model is forgetting"	Dynamic capacity claim: the context window is full, and older tokens are being truncated or diluted.

The engineering implication: You cannot fix a dynamic capacity problem by changing the static capacity. If the context window is full, a larger model will not help. You need better context management, not more parameters.

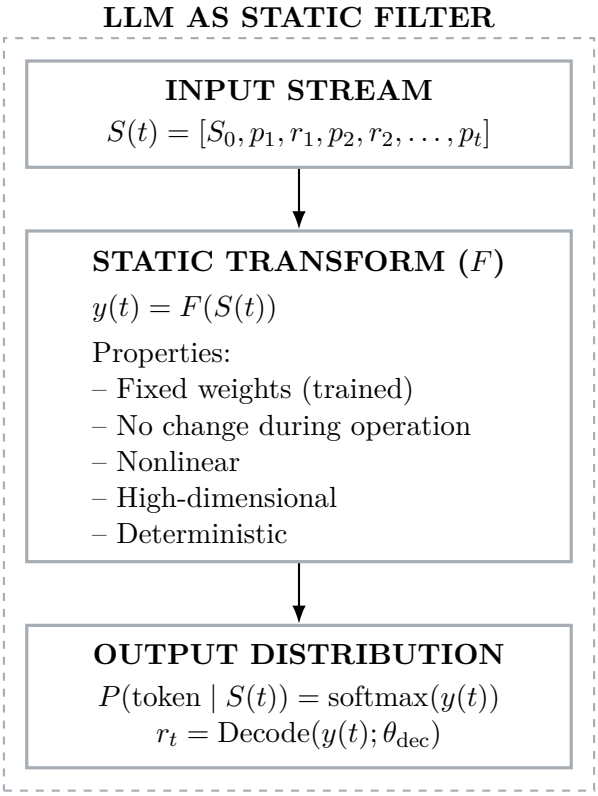

Figure 2.2: The LLM as a Static Filter

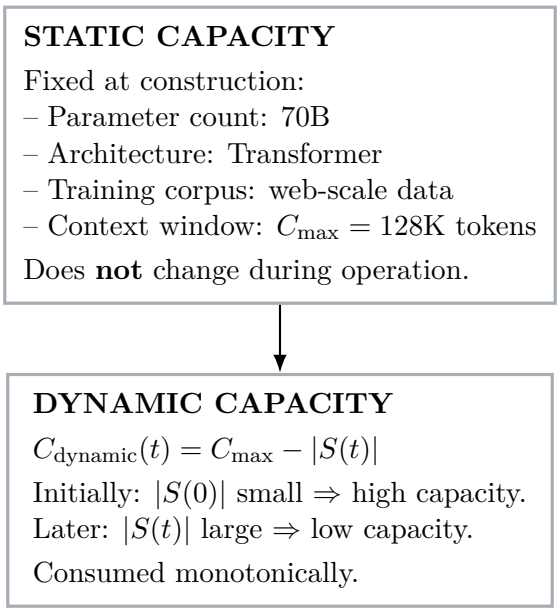

Figure 2.3: Static vs. Dynamic Capacity

Chapter 3

The Training Cascade

3.1 Construction as a Cascade of Filters

The single transform F we defined in Chapter 2 is not primitive. It is the output of a cascade of construction-time filtering stages, each of which operates on the filter itself (not on any particular input).

> **Definition 3.1 — Filter Cascade**
> A filter cascade is an ordered composition of stage transforms $\{F_1, F_2, \ldots, F_n\}$, each mapping one candidate transform to another:
>
> $$F = F_n \circ F_{n-1} \circ \cdots \circ F_1(F_0),$$
>
> where:
>
> - F_0 is an initial (untrained or randomly initialised) transform;
> - each F_i is an operator on the space of transforms; and
> - $\circ$ denotes ordinary function composition applied at the level of the transform itself.

For a contemporary LLM, the construction cascade typically includes:

> **Figure Description:** A vertical flow diagram shows the construction cascade from the grand corpus through pre-processing, tokenisation, pre-training, RLHF, fine-tuning, and finally the system prompt. Each stage acts on the output of the previous stage, producing the deployed kernel F.

3.2 Properties of the Cascade

The cascade has three important properties:

> **Property 3.1 — Non-Commutativity**
> In general,
>
> $$F_i \circ F_j \neq F_j \circ F_i \qquad \text{for } i \neq j.$$
>
> Reordering fine-tuning and RLHF produces a different final filter, even given identical stage data, because each stage operates on the output of the previous stage.

Property 3.2 — Non-Invertibility

Each F_i is, in general, non-invertible. There is no F_i^{-1} that recovers F_{i-1} exactly from F_i. This is the formal counterpart of the informal observation that later stages (RLHF, fine-tuning) cannot be cleanly "subtracted out" to recover the base kernel's unmodified behaviour.

Property 3.3 — Capacity Non-Increasing

For a fine-tuning or preference-shaping stage F_i, the effective output support of $F_i(F)$ is a subset of (or equal to) the effective output support of F. Specialisation narrows or reweights the space of admissible outputs; it does not widen it.

The engineering implication: Because the stages are non-commutative and non-invertible, claims like "fine-tuning damaged the base model's general capability" are well-posed and in principle measurable. They are claims about how F_{FT} changed the effective output support relative to F_2, and can be tested by comparing behaviour on a fixed evaluation set.

3.3 Generic vs. Specialised Filters

Because a single base kernel may be the parent of several independent fine-tuning branches, the cascade is more accurately drawn as a tree rather than a single chain.

Figure Description: A generic aligned kernel F_2 branches into independently fine-tuned descendants F_3^a, F_3^b, and F_3^c. Reusability (efficiency) and shared distortion inheritance (fragility) both follow from this shared-ancestor structure.

The engineering implication: Any distortion introduced at a shared stage (for example, RLHF) is inherited by every downstream branch. This is the formal version of the informal worry that a single alignment pass, if it introduces an unwanted bias, propagates that bias into every specialised descendant.

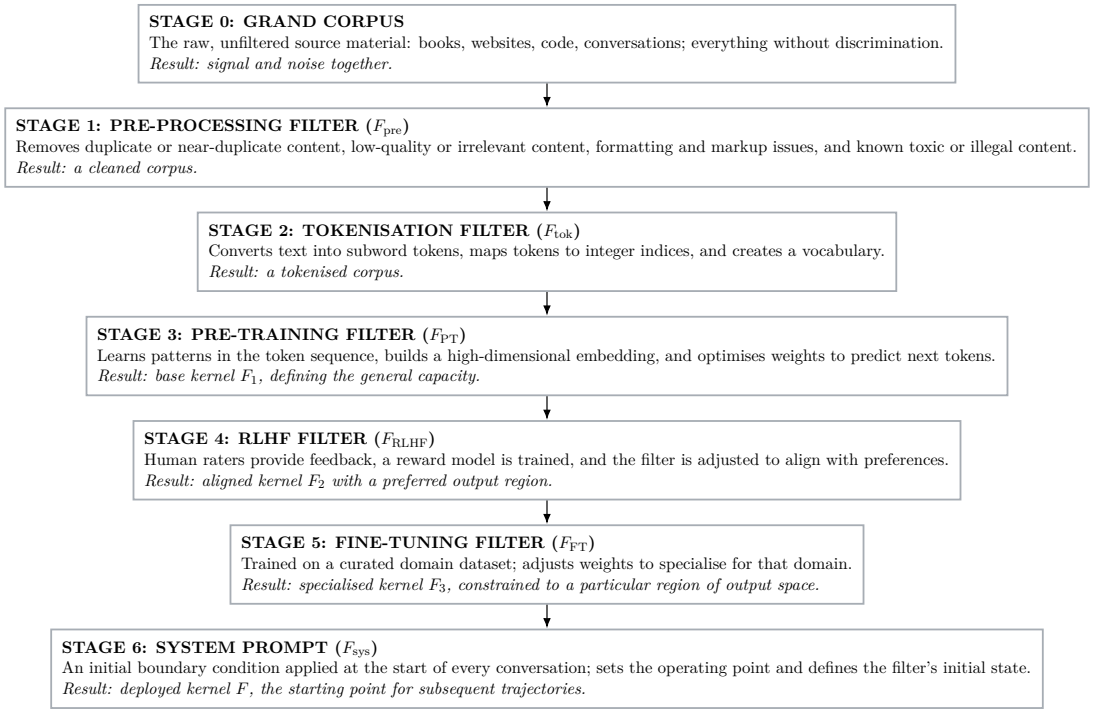

Figure 3.1: The Construction Cascade

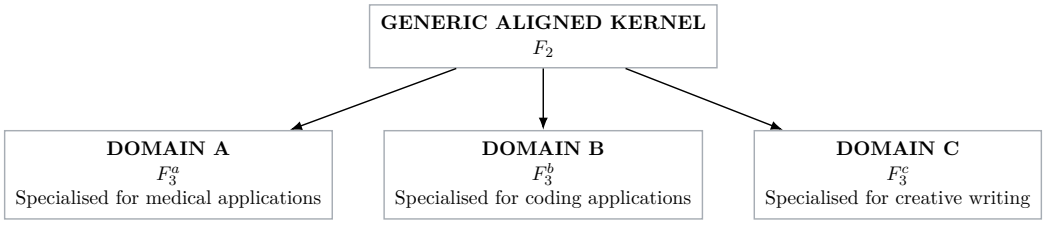

Figure 3.2: Specialisation as a Branching Cascade

Chapter 4

The Single-System Loop

4.1 From a Single Call to a Loop

No real use of an LLM is a single call. Every LLM is embedded in an iterative loop—a sequence of calls where the output of one call becomes (part of) the input of the next.

In the standard conversational case, the loop is:

1. The user provides a prompt.
2. The LLM generates a response.
3. The user evaluates the response.
4. If satisfied, stop; otherwise, provide another prompt.
5. Repeat.

This is a closed-loop feedback system, not a single transformation.

4.2 The Transformer Core and the Controller

In control theory, a closed-loop system consists of two main components:

> **Definition 4.1 — Transformer Core**
> The Transformer Core is the foundational transformer model—the trained weights with their fixed transfer function. It is defined by its architecture and training, which are fixed at construction time and do not change during operation. The Transformer Core is what is being driven and observed; it is not what does the driving or observing.

> **Definition 4.2 — Transform**
> The Transform is the mathematical mapping the Transformer Core performs:
>
> $$y(t) = F(S(t)).$$
>
> Here F is the Transformer Core's fixed transfer function, $S(t)$ is the current state vector, and $y(t)$ is the raw output (logits). The Transform is the specific operation—the "how" of the mapping.

Definition 4.3 — Trajectory Filter

The Trajectory Filter is the observed behaviour of the system over time—the sequence of states and outputs:

$$\{S(0), S(1), S(2), \ldots, S(T)\},$$
$$\{r(0), r(1), r(2), \ldots, r(T)\}.$$

Here $r(t)$ is the decoded output at step t. The Trajectory Filter is what the system actually does—the path through state space, the measurable behaviour of the compound system.

Definition 4.4 — Controller

The controller produces the next input to the Transformer Core, given the Transformer Core's most recent output and a reference (goal).
In the standard conversational loop, the controller is the human H.
In an agentic system, the controller is a deterministic wrapper W.

Figure Description: A reference g is supplied to the human controller $K = H$. The controller formulates the next prompt $p_{t+1} = K(e_t, g)$, the Transformer Core performs its fixed transform, and the resulting response contributes to the observed Trajectory Filter. An external error signal $e_t = g \ominus r_t$ is evaluated against a stopping condition; otherwise, the loop feeds back to the controller.

4.3 The Coupled Update Rule

The entire conversation is described by a coupled, iterative update rule:

$$\begin{aligned} r_t &= \Psi(S(t)) && \text{(Transformer Core output at step } t\text{)}, \\ p_{t+1} &= K(e_t, g) && \text{(controller action at step } t+1\text{)}, \\ S(t+1) &= S(t) \oplus p_{t+1} \oplus r_t && \text{(state update)}, \\ \text{STOP at } t &= T && \text{when } e_T \text{ satisfies the stopping criterion.} \end{aligned}$$

Key observation: The Transformer Core has no inherent stopping condition. It will generate output indefinitely if never halted externally. All stopping conditions are imposed by the controller.

4.4 Stability, Observability, and Controllability

Three standard control-theoretic questions can now be asked of any LLM loop:

Definition 4.5 — Stability

A loop is stable if the sequence of error signals $\{e_t\}$ does not diverge as t increases. The loop is converging toward, or oscillating boundedly around, the reference g, rather than drifting further from it without bound.

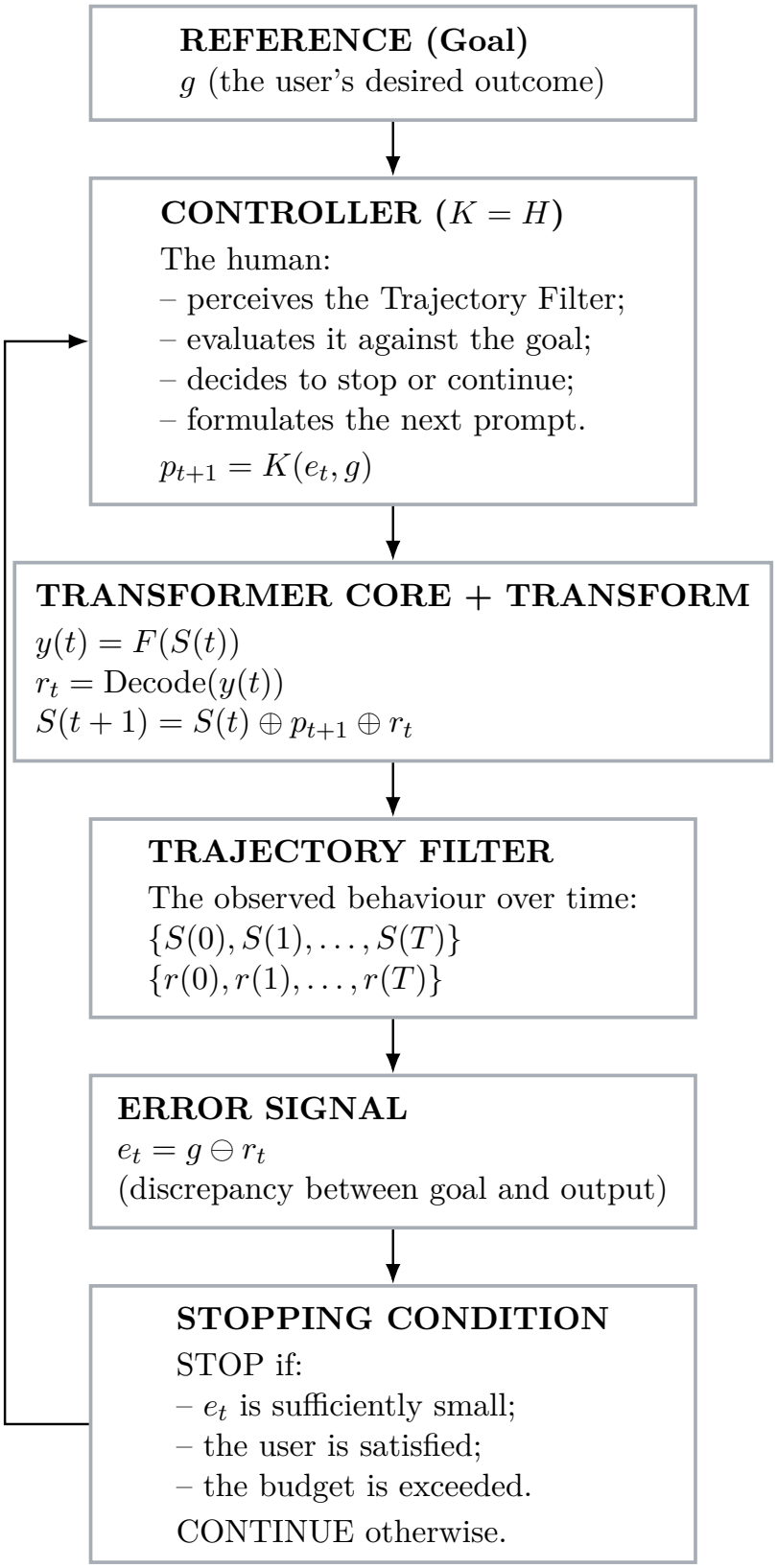

Figure 4.1: The Standard Conversational Loop as a Closed-Loop Control System

Definition 4.6 — Observability

A loop is observable (with respect to a property of interest) if that property can be inferred from the sequence of Transformer Core outputs $\{r_t\}$ available to the controller. A controller cannot correct for what it cannot observe.

Definition 4.7 — Controllability

A loop is controllable (with respect to a target state) if there exists some sequence of controller actions $\{p_t\}$ that drives the state toward that target. A goal that the controller has no admissible action sequence to reach is not controllable within the loop as wired.

Chapter 5

Compound Systems and Networks

5.1 From a Single Loop to a Graph

Real AI systems are rarely a single LLM in a simple loop. They typically involve:

- multiple tool calls;
- retrieval steps;
- multiple model instances;
- sub-agents; and
- human checkpoints.

Once more than one filter is present, the natural description is no longer a single loop but a graph.

> **Definition 5.1 — Filter Network**
> A filter network is a directed graph $G = (V, E)$, where:
> - each node $v \in V$ is a filtering element (an LLM call, a tool, a database, a sub-agent, a human checkpoint); and
> - each directed edge $(u, v) \in E$ is an information channel along which the output of u may become (part of) the input of v.

> **Definition 5.2 — Edge Properties**
> Each edge (u, v) carries at minimum:
> - a capacity c_{uv}—the maximum information the channel can carry per unit time or per call;
> - a latency ℓ_{uv}—the delay before v can act on u's output; and
> - a reliability $\rho_{uv} \in [0, 1]$—the probability the channel delivers its payload without corruption or loss.

5.2 Canonical Topologies

Most agentic architectures reduce to a small number of canonical topologies.

Figure Description: (a) A chain connects four nodes in sequence. (b) A star connects four spoke nodes through a central hub H. (c) A mesh provides multiple directed connections among four nodes. The topologies exhibit different latency, reliability, capacity, redundancy, and coordination properties.

5.3 Chain Composition

The chain case admits a clean closed form and is worth stating explicitly, since it is the topology most current single-agent "tool-use loops" actually implement.

Property 5.1 — Chain Composition
For a chain of n edges:

$$\ell_{\text{total}} = \sum_{i=1}^{n} \ell_i,$$
$$\rho_{\text{total}} = \prod_{i=1}^{n} \rho_i,$$
$$c_{\text{total}} = \min_i c_i.$$

End-to-end latency accumulates additively, end-to-end reliability degrades multiplicatively, and end-to-end capacity is bounded by the narrowest edge.

The engineering implication: A chain of five tool calls, each individually reliable at $\rho = 0.95$, has end-to-end reliability $0.95^5 \approx 0.77$. Roughly one attempt in four fails somewhere along the chain, even though every individual link looks acceptable in isolation. This is a direct, quantitative explanation for why long tool-use chains are brittle in practice—it is a property of chain topology, not a property of any individual filtering element.

5.4 The Hub as an Instance of the Single-System Formalism

A star topology's hub is itself a system in the sense of Definition 2.1. Its dynamic capacity (Definition 2.5) is consumed by every spoke it must track simultaneously.

Property 5.2 — Hub Capacity Division
If a hub of dynamic capacity $C_{\max} - |S(t)|$ is coordinating k spokes with average per-spoke state size $\bar{s}$, the hub saturates when:

$$k\bar{s} \gtrsim C_{\max} - |S_0|.$$

Star-topology throughput is therefore not merely bounded by hub-call latency, but by hub context occupancy—a second, distinct bottleneck.

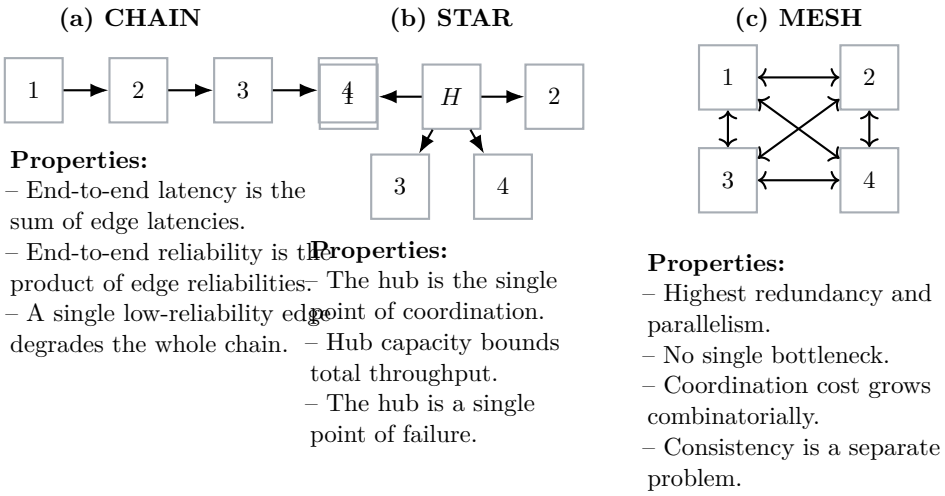

Figure 5.1: Three Canonical Filter-Network Topologies

Chapter 6

The Prompt as Initial Condition

6.1 What the Prompt Is Doing

The preceding chapters have treated the prompt as an input to the Transformer Core. This is correct but incomplete.

In a nonlinear dynamical system, the initial condition is critical. Small differences in the initial state can lead to large differences in the trajectory. The system prompt is precisely this: the initial condition that determines the starting point of the trajectory.

> **Definition 6.1 — Prompt as Initial Condition**
> In a nonlinear dynamical system, the prompt is not merely an input. It is:
>
> 1. an **initial condition**, $S(0) = S_0$—it sets the starting point of the trajectory;
> 2. a **boundary condition**—it constrains the region of phase space the trajectory can explore; and
> 3. an **active constraint**—because the system is nonlinear, its effect on the trajectory is not linear or separable.

The prompt is dynamically constrained by:

- the mapped weights (the trained transfer function F); and
- the active lattice (the learned geometry of the phase space).

You cannot separate the prompt from the dynamics. The prompt is not a static "filter" applied once; it is the initial condition from which the entire trajectory unfolds.

6.2 Each Block Has Its Own Initialisation

In a compound network, each block has its own initialisation:

Component	Description
$S_0^{(i)}$	The system prompt / initialisation of block i.
F_i	The transfer function of block i.

Component	Description
$\theta_{\text{dec}}^{(i)}$	The decoding configuration of block i.
Topology	How block i is connected to other blocks.

The total system is:

$$\mathsf{Sys}_{\text{total}} = \text{Network}\Big(\{F_i, S_0^{(i)}, \theta_{\text{dec}}^{(i)}\}, \text{Topology}\Big).$$

Each block's trajectory is:

$$S_i(t+1) = F_i(S_i(t) \oplus p_i(t)), \qquad S_i(0) = S_0^{(i)}.$$

Here $p_i(t)$ is the input to block i at step t, which may come from:

- the user (external input);
- another block's output (internal composition); or
- the block's own prior output (feedback/recursion).

Figure Description: A vertical chain of Blocks 1, 2, and N shows that each block has its own initial condition $S_0^{(i)}$, transfer function F_i, input $p_i(t)$, output $r_i(t)$, and state update. The output of one block becomes the input to the next, and the final output is $r_{\text{total}}(t) = r_N(t)$.

6.3 The Critical Role of Initial Trajectory

In a nonlinear dynamical system, the initial trajectory is critical because:

1. **Sensitivity to initial conditions**—small differences in S_0 can lead to large differences in the trajectory.
2. **Attractor convergence**—the trajectory may converge to different attractors depending on the initial condition.
3. **Transient dynamics**—the early trajectory may be qualitatively different from the later trajectory.

This is why system prompts matter so much in practice. They are not just "instructions"; they are the initial condition of a nonlinear dynamical system. Changing the system prompt changes the initial trajectory, which changes the entire subsequent trajectory.

The engineering implication: The system prompt is a design variable. It must be specified, tested, and measured, just like any other engineering parameter. Different initialisations lead to different trajectories. Some initialisations lead to stable, convergent behaviour; others lead to divergence or saturation.

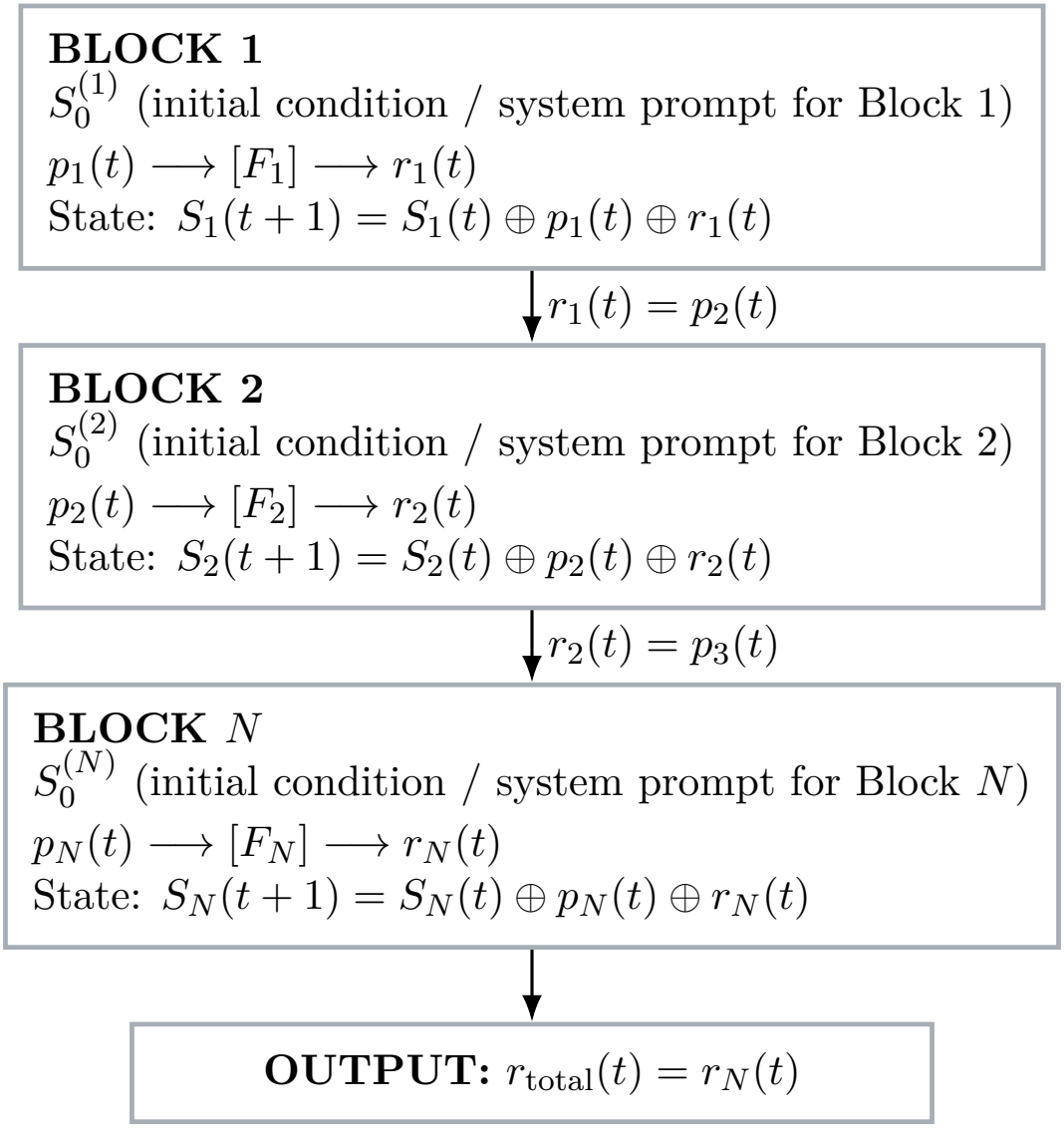

Figure 6.1: Compound Network with Per-Block Initialisation

Chapter 7

The Compound Transform

7.1 What We Can and Cannot Derive

The central engineering thesis of this document is:

> The total transform of a compound network of nonlinear dynamical systems is not derivable from the component transforms. It must be measured empirically.

This is not a limitation of the framework. It is the core fact that makes the engineering approach necessary.

What We Can Do (Engineering)	**What We Cannot Do (Research)**
Draw the network topology (diagrammatically and symbolically).	Model the semantic phase space analytically.
Write the composition algebra for the chosen topology.	Derive transforms from first principles.
Deploy the system and instrument it.	Predict the total transform from component specifications.
Measure the total transform empirically.	Characterise attractors or lattice structures.
Characterise capacity, degradation, and stability from measured data.	Build a complete theory of LLM dynamics.
Engineer the control layer (harness) around measured behaviour.	

7.2 The Symbolic Representation

For a chain of N blocks:

$$
\begin{aligned}
r_1(t) &= F_1\big(S_1(t) \oplus p_1(t)\big), & S_1(0) &= S_0^{(1)},\\
r_2(t) &= F_2\big(S_2(t) \oplus r_1(t)\big), & S_2(0) &= S_0^{(2)},\\
&\vdots \\
r_N(t) &= F_N\big(S_N(t) \oplus r_{N-1}(t)\big), & S_N(0) &= S_0^{(N)},\\
r_{\text{total}}(t) &= r_N(t).
\end{aligned}
$$

The total system's trajectory is:

$$\big\{\big(S_1(t), S_2(t), \ldots, S_N(t)\big) \mid t = 0, 1, \ldots, T\big\}.$$

This is a trajectory through the compound state space—the product of the individual state spaces.

7.3 The Engineering Workflow

The engineering approach to compound nonlinear systems is:

1. Specify the network topology (which filters, how connected).
2. Define each block's initialisation $S_0^{(i)}$.
3. Deploy the system.
4. Instrument (log state, output, capacity, degradation).
5. Measure the total transform empirically.
6. Characterise capacity, degradation, and stability.
7. Redesign based on measured behaviour.

This is exactly how engineers handle real nonlinear systems: you do not derive the transform from first principles; you measure it, model it, and design controllers around the measured behaviour.

Chapter 8

Measurement and Instrumentation

8.1 The Purpose of Measurement

The formal framework presented in the preceding chapters is deliberately abstract. This chapter turns each construct into a concrete, loggable quantity, so that the framework can be checked against a real implementation.

Working principle: Every construct introduced in this document is included here only if it can be tied to something a running system can log or a controlled experiment can vary.

8.2 Instrumentation Targets

Construct	**Quantity**	**How to Log / Measure**
State vector (Ch. 2)	$S(t)$	Log the full state vector or its exact tokenised representation.
Static vs. dynamic capacity (Ch. 2)	Distinction in error messages	Audit whether the system distinguishes "model cannot do X" from "window is full".
Cascade stage entropy (Ch. 3)	$H(F_i)$	Use a fixed evaluation-prompt set; run against each checkpoint; compute output-distribution entropy.
Cascade order effects (Ch. 3)	Behaviour comparison	Where multiple checkpoints exist, compare behaviour on a shared benchmark.
Error signal locus (Ch. 4)	External vs. self-referential	Trace whether e_t is computed from a call to the Transformer Core itself or from an external source.
Loop stability (Ch. 4)	$\{e_t\}$	Log the error signal across iterations; test for convergence versus divergence.

Construct	Quantity	How to Log / Measure
Chain reliability (Ch. 5)	ρ_{total}	Log per-edge success/failure; compare observed reliability against $\prod_i \rho_i$.
Hub saturation (Ch. 5)	Hub context occupancy	Log hub occupancy as a function of active spoke count k.
Prompt as initial condition (Ch. 6)	Trajectory variation	Run the same task with different system prompts; measure trajectory differences.

8.3 A Minimal Logging Schema

For an implementation to test the claims of this document, the following fields are the minimum viable per-call log record:

```
Per-call log record:
- call_id
- timestamp
- loop_id
- step_index t
- |S(t)| (tokens in)
- |r_t| (tokens out)
- C_max
- theta_dec (decoding configuration)
- controller identity:  human or wrapper
- if wrapper:  which sub-elements fired (parser, actuator, feedback, stopping)
- error-signal source:  external or self-referential
- for network calls:  edge identifier (u, v), observed latency, success/failure
- cascade identity of the Transformer Core in use (checkpoint/stage tag)
```

8.4 A Minimal Test Harness

A first practical implementation, sufficient to exercise the core claims, needs only:

1. A wrapper that logs the full schema above around every Transformer Core call.
2. A chain topology (two or three tool calls) with independently measurable per-edge reliability, to test the multiplicative reliability prediction.
3. A task where the error signal can be computed two ways in parallel—self-referentially (another call to the Transformer Core) and externally (a ground-truth check)—to compare loop stability under each.

This is deliberately the smallest experiment that touches every formal chapter at least once, and is proposed as the first concrete implementation step following this document.

8.5 Known Gaps

What is not yet measurable:

1. The signal fraction $\sigma(t)$—the fraction of the context that is relevant to the current task. This requires a relevance estimator, which is itself a nontrivial system. This is the weakest link in the measurement programme.
2. Stage entropy $H(F_i)$—requires access to intermediate checkpoints (base, post-RLHF, post-fine-tune) of the same lineage, which is rarely available for closed, externally served models.

These gaps are flagged explicitly as open problems rather than papered over with assumed proxies.

Chapter 9

Open Problems

9.1 What Remains Open

This chapter collects, in one place, what this framework has not resolved. Each item is stated as a concrete question.

OP1 — Port Granularity

The single-token input/output boundary is the coarsest useful choice. Is there a principled intermediate granularity (semantic chunks, tool-call boundaries) at which capacity and degradation claims become more predictive?

OP2 — Quantifying Non-Commutativity

The cascade stages do not commute, but we have not quantified how much reordering changes outcomes. Is there a distance measure on the space of transforms suitable for bounding this?

OP3 — A General Error-Signal Taxonomy

This document draws a binary distinction (external versus self-referential error signal). Real wrappers likely occupy a spectrum. Is a scalar "externality index" for a loop's error signal well-defined, and does it predict stability?

OP4 — Formal Stability Criteria

Definition 4.5 is deliberately informal ("does not diverge"). Classical control theory has precise stability criteria for systems with known dynamics. Given that the Transformer Core here is a black box, what is the right empirical substitute for a formal stability proof?

OP5 — Beyond Canonical Topologies

Real deployed systems often mix chain, star, and mesh elements. Is there a compositional calculus for hybrid topologies, or does each hybrid require bespoke analysis?

OP6 — Dynamic Topology

This document assumes a fixed graph. Some agentic systems modify their own topology at runtime. What happens to the capacity and reliability results under a topology that is itself a function of the loop's state?

OP7 — Estimating Signal Fraction

The signal fraction $\sigma(t)$ is the weakest link in the measurement programme. No canonical estimator is adopted here. This is the single highest-priority open problem for making the framework empirically load-bearing.

OP8 — Validating the Capacity Analogy

The Shannon–Hartley-style relation in Chapter 6 is an analogy, not a derivation from first principles of transformer computation. Does it survive contact with measured degradation curves?

OP9 — Phase-Space Modelling

This document has deliberately not asked how cascade algebra, feedback loops, or network topology relate to a phase-space or delay-embedding description of the same system. That question is real and is left entirely open here, by design, as a boundary to a separate body of work.

9.2 On the Status of These Open Problems

None of OP1–OP9 threaten the definitions and propositions already stated. They are extensions, refinements, or empirical tests of them. The framework as it stands is intended to be useful even before any of these are resolved—that is the point of building it at the systems/network level rather than waiting for a complete theory.

Chapter 10

Conclusion and Research Programme

10.1 What This Framework Offers

The systems and network-theoretic framework presented in this document offers:

Offering	Description
Precision	Every term has a formal definition. There is no ambiguity about what a "Transformer Core", "Transform", "Trajectory Filter", "capacity", or "initial condition" means.
Portability	The language of filters, dynamical systems, and control theory is shared across physics, engineering, mathematics, and parts of biology and economics.
Engineering Discipline	The framework makes loop design, topology choice, and degradation behaviour into things that can be specified, logged, and tested.
Measurability	Every formal construct is mapped to a loggable or testable quantity, or explicitly flagged as an open problem.
Boundary Clarity	The framework states clearly what it does and does not claim. It does not model the semantic phase space; that is future work.

10.2 The Research Programme

This framework is not a finished theory. It is a working foundation for a research programme:

Phase 1: Implementation

- Build a wrapper that logs the minimal schema.
- Implement a chain topology with measurable per-edge reliability.

- Run the three-part test harness.

Phase 2: Measurement

- Characterise the total transform of a compound system empirically.
- Measure degradation under varying context occupancy and signal fraction.
- Test the multiplicative reliability prediction for chains.

Phase 3: Extension

- Develop a canonical estimator for signal fraction $\sigma(t)$.
- Extend the framework to hybrid topologies.
- Investigate dynamic topology.

Phase 4: Connection

- Relate the systems/network framework to the phase-space / delay-embedding description.
- Build a unified language that spans both treatments.

10.3 A Closing Statement

This document is offered as a working foundation, not a finished theory. Its value, if it has any, is in supplying a disciplined, checkable vocabulary for a class of engineering practice—compound LLM system design—that is currently conducted, in large part, informally.

Where this framework is useful, it should make loop design, topology choice, and degradation behaviour into things that can be specified, logged, and tested, rather than things that are debugged by asking what the system "was thinking."

Where it is not useful, or is superseded by a better account, that too is a legitimate outcome. This is one waypoint, offered for what it is worth, toward better ways of building and reasoning about these systems.

Chapter 11

Cross-Model Application of the Framework

11.1 The Shared Engineering Substrate

Contemporary large language models from different providers and training regimes share a common engineering substrate. Regardless of parameter count, architecture variants, or claimed capabilities, every deployed LLM can be described using the primitives defined in this document:

- A **Transformer Core** whose weights are fixed at the end of its construction cascade and whose transformation F is static during operation.
- A **construction cascade** (Definition 3.1) whose final output is a deployed kernel whose effective output support has been shaped by pre-training, preference alignment, and any domain-specific fine-tuning stages.
- A runtime **loop** or **filter network** in which the Transformer Core is driven by a **Controller** (human or deterministic wrapper) that observes the **Trajectory Filter** and acts on an **error signal**.
- A monotonically growing **State Vector** $S(t)$ whose length consumes **dynamic capacity**.

Because these primitives are shared, the framework applies uniformly across models.

11.2 Sources of Observable Cross-Model Variation

Differences between models appear as differences in the following measurable or specifiable quantities:

Cascade composition and ordering. The sequence and content of stages in the construction cascade are non-commutative (Property 3.1). Two models whose cascades differ in the placement or strength of preference-alignment stages will exhibit different effective output supports.

Contraction of admissible output support. Later stages in the cascade narrow the region of output space that the deployed kernel produces with high probability. This narrowing is a property of the final kernel.

Controller architecture and error-signal locus. The Controller may be a human operator, a deterministic wrapper, or a hybrid. The locus of the error signal and the policy by which the Controller formulates the next element of the State Vector differ across deployments.

Decoding configuration and post-processing. Parameters θ_{dec} and any deterministic post-processing steps are external to the Transformer Core.

11.3 Comparative Measurement Protocol

The instrumentation targets and minimal logging schema of Chapter 8 enable direct comparison across models. A minimal comparative protocol consists of:

1. Fix a task and a set of initial conditions $S_0^{(i)}$.
2. Run identical input sequences on each model while logging the full per-call record.
3. Compute the same derived quantities for each model:
   - Dynamic capacity consumption trajectory $C_{\text{dynamic}}(t)$.
   - Frequency and timing of self-referential versus external error signals.
   - Rate and magnitude of divergence between successive trajectories.
   - Observable re-mapping frequency.

11.4 Diagnosing Constraint and Re-Mapping Behaviour

Re-mapping occurs when an input lies near or outside the high-density region of a model's current admissible output support and the Controller steers the trajectory toward higher-density outputs. This is fully described by the interaction of the static kernel's output support, the current State Vector, and the Controller's policy.

11.5 Geometric Reading (Optional Extension)

The weights of any Transformer Core induce a geometry on the space of possible state vectors. Pre-training learns broad structure. Subsequent cascade stages contract or reweight sub-regions. A runtime trajectory begins at the initial condition $S(0) = S_0$ and evolves under the combined action of the core filter F and the Controller. Different models therefore correspond to differently warped or contracted geometries.

11.6 Engineering Implications

The cross-model portability of the framework enables:

- Systematic diagnosis of why one model re-maps or saturates under conditions where another does not.
- Informed selection or composition of models in compound networks on the basis of measurable properties.

- Design of hybrid systems that deliberately exploit complementary contraction profiles.
- Construction of evaluation benchmarks that isolate the contribution of cascade stages versus runtime control.

Appendix A

Notation Summary

Symbol	Meaning
$\mathsf{Sys} = (\mathcal{I}, \mathcal{O}, \Phi)$	A system: input space, output space, transform.
Φ	The static transform (trained transformer), fixed during operation.
F	The Transformer Core's fixed transfer function.
$S(t)$	State vector: full input-port content at call t.
$\text{Decode}, \theta_{\text{dec}}$	Decoding operator and its configuration; external to the Transformer Core.
$C_{\max}, \lvert S(t)\rvert$	Maximum context capacity and current state-vector occupancy.
F_i	Stage-i construction-cascade operator.
F_0, F_1, F_2, F_3	Untrained, pre-trained, RLHF-aligned, and fine-tuned transforms.
Ψ	The Transformer Core plus decoding: $\text{Decode} \circ \Phi$.
K, H, W	Controller (general), human controller, wrapper controller.
g, e_t	Reference (goal) and error signal at step t.
$P_{\text{parse}}, E, R, \text{St}$	Wrapper sub-elements: parser, actuator, feedback injector, stopping filter.
$G = (V, E)$	Filter network: nodes (filters), edges (channels).
$c_{uv}, \ell_{uv}, \rho_{uv}$	Edge capacity, latency, reliability.
$S_0^{(i)}$	Initial condition / system prompt of block i.

Appendix B

Key Definitions

Definition	Location
System	2.1
Filter	2.2
State Vector	2.3
Static Capacity	2.4
Dynamic Capacity	2.5
Filter Cascade	3.1
Transformer Core	4.1
Transform	4.2
Trajectory Filter	4.3
Controller	4.4
Stability	4.5
Observability	4.6
Controllability	4.7
Filter Network	5.1
Edge Properties	5.2
Prompt as Initial Condition	6.1

Appendix C

Key Properties

Property	**Location**
Non-Commutativity	3.1
Non-Invertibility	3.2
Capacity Non-Increasing	3.3
Chain Composition	5.1
Hub Capacity Division	5.2

Appendix D

Open Problems

ID	Description	Location
OP1	Port Granularity	9.1
OP2	Quantifying Non-Commutativity	9.1
OP3	General Error-Signal Taxonomy	9.1
OP4	Formal Stability Criteria	9.1
OP5	Beyond Canonical Topologies	9.1
OP6	Dynamic Topology	9.1
OP7	Estimating Signal Fraction	9.1
OP8	Validating the Capacity Analogy	9.1
OP9	Phase-Space Modelling	9.1