SKYTEAM AEROSPACE FOUNDATION

Research and Development
Wynne, AR

CBSZ-HH **December 15, 2025**

Architectural Skeleton of the Dynamic Transformer Architecture (DTA)

Proposed By: Zenith Zaraki

Abstract

Current large language models (LLMs) function as stateless autoregressive predictors. Their internal computations reset at every token, providing no persistent latent state that evolves across a sequence. This structural limitation leads to well-documented failure modes, including representational drift, instability under noise, loss of long-range coherence, inconsistent reasoning, and unpredictable degradation under runtime stress.

This manuscript proposes the Dynamic Transformer Architecture (DTA), a theoretical framework designed to address these limitations while maintaining full compatibility with existing Transformer infrastructures. DTA augments a standard Transformer with a parallel Dynamic State Path (DSP) that computes five internal stability signals at each generation step:

- **Internal Coherence (I)** — representational consistency between the evolving latent state and current hidden representations
- **Computational Alignment (C)** — directional agreement between instantaneous hidden-state dynamics and accumulated internal state
- **Dynamic Noise (N)** — estimate of destabilizing representational turbulence
- **Substrate Stability (S)** — runtime-conditioned measure of hardware or execution reliability
- **Temporal Continuity (T)** — stability of the evolving latent trajectory across tokens

These quantities are combined through a Recursive Integration Operator (RIO) derived from a general state-evolution formulation:

$$s_{t+1} = S_t \times T_t \times \left(\alpha s_t + (1-\alpha)\frac{I_t C_t}{N_t}\right),$$

where s_t is the internal state vector maintained across token generation.

This recurrence introduces persistent latent dynamics into a Transformer in a mathematically governed, stability-regulated manner. It enables:

- long-horizon representational continuity,
- reduced drift and improved internal consistency,
- predictable behavior under destabilizing or noisy input,
- controlled degradation under substrate stress,
- and stable recurrence across extended sequences.

All of this is achieved without modifying the Transformer's attention mechanism, projection heads, positional encoding, or inference pipeline. DTA operates as a drop-in augmentation compatible with PyTorch, ONNX, CUDA, DirectML, inference servers, and existing KV-cache methods.

As a theoretical architecture presented for research and exploratory analysis, DTA represents a transitional framework between today's stateless Transformers and future state-regulated models capable of stable, long-range reasoning on real-world workloads.

Section 1 — Architectural Skeleton of the Dynamic Transformer Architecture (DTA)

The Dynamic Transformer Architecture (DTA) is a proposed framework consisting of three coordinated computational pathways designed to operate in parallel during inference. Together, they introduce persistent state evolution, stability regulation, and internal-dynamics monitoring while preserving full compatibility with standard Transformer infrastructures. Although presented here in theoretical form, the architecture is structured to be implementable within existing inference systems.

1.1 Transformer Path (TP)

The Transformer Path is the unmodified attention-based sequence processor used in existing autoregressive models (GPT-family, LLaMA-family, etc.). It performs:

- token embedding and projection
- multi-head self-attention
- feed-forward transformations
- residual connections and normalization

DTA leaves this pathway untouched to ensure:

- full backward compatibility with pretrained models
- interoperability with existing inference engines (PyTorch/ONNX/CUDA/DirectML)
- preservation of standard KV-cache mechanisms
- unaltered next-token prediction behavior

The TP remains responsible for the primary linguistic and representational computation.

1.2 Dynamic State Path (DSP)

The Dynamic State Path introduces the persistent, evolving internal state vector and computes the internal stability signals required to regulate the model's long-range behavior.
In the proposed design, the DSP carries state from token to token and evaluates the internal conditions under which that state should evolve.

The DSP computes:

- **Temporal State (TPC):** a trainable recurrent state vector updated each token
- **State-Evolution Signal:** application of the DTA state-update equation over time
- **Noise Estimate (N):** real-time measurement of representational instability
- **Alignment Score (C):** directional similarity between hidden-state dynamics and persistent state
- **Substrate Stability (S):** runtime-conditioned signal reflecting hardware reliability
- **Internal Coherence (I):** scalar measure of consistency between hidden and state trajectories

These components give the architecture:

- persistent state continuity
- controlled evolution across long sequences
- resistance to drift under noise
- predictable degradation under runtime stress
- internal monitoring of representational quality

Classical Transformers lack all of these capabilities.

1.3 Integration Path (IP)

The Integration Path merges the Transformer Path and the Dynamic State Path outputs at each generation step.

Its responsibilities include:

- injecting the evolving state vector into Transformer hidden states
- biasing token generation toward stable, low-noise, high-coherence trajectories
- reinforcing representational continuity across tokens
- suppressing instability by modulating hidden-state dynamics before projection

Formally, the IP applies a residual correction to the Transformer's hidden representation:

$$h_t^{\text{final}} = h_t + W\, s_t$$

where:

- $h_t$is the Transformer hidden state
- $s_t$is the DTA internal state vector
- $W$is a learned projection matrix

This mechanism preserves the Transformer architecture while enabling state-informed generation.

1.4 Why This Architecture Works

DTA succeeds because of its **strict separation of concerns**:

1. **The Transformer Path**
    - retains the full expressive power, efficiency, and backward compatibility of existing models.
    - remains unchanged, avoiding any reengineering of attention mechanisms, positional encoding, or KV-cache logic.
2. **The Dynamic State Path**
    - supplies the missing engineering components:
      persistent state, continuity dynamics, noise modeling, alignment measurement, and state-regulated evolution.
3. **The Integration Path**
    - couples the two systems without interfering with normal Transformer computations or breaking runtime infrastructure.

In effect:

- The Transformer Path provides sequence prediction.
- The Dynamic State Path provides persistent state dynamics and internal-signal regulation.
- The Integration Path fuses these signals into the generation loop.

This division allows the proposed DTA framework to act as a drop-in enhancement to existing models while introducing the structural components required for stable, long-horizon, state-dependent computation.

Section 2 — Temporal Persistence Core (TPC)

2.1 Overview

The Temporal Persistence Core (TPC) is a proposed mechanism for providing the token-to-token state continuity required for the Dynamic Transformer Architecture (DTA) to function as a stateful sequence model. Traditional Transformers operate as stateless, feed-forward predictors: each output depends only on the local hidden representation and the KV-cache, not on a persistent internal state representing system dynamics over time. This constraint leads to:

- representational drift across long sequences,
- instability under noise,
- loss of directional consistency,
- weak long-range control signals.

The proposed TPC introduces a persistent, trainable state vector that would evolve during generation. This state encodes the model's internal dynamical trajectory, allowing DTA to regulate:

- representational continuity,
- internal stability,
- drift suppression,
- long-horizon behavior.

In the overall framework, the TPC is the core mechanism that conceptually converts a stateless Transformer into a state-evolving dynamical system.

2.2 Architectural Component

At token timestep t, the Temporal Persistence Core is defined to update as:

$$\mathrm{TPC}_t = \mathrm{GRU}([h_t, s_t, r_t], \mathrm{TPC}_{t-1}),$$

where:

- h_t— Transformer hidden state at token t
- s_t— DTA internal state vector at token t
- r_t— Transformer residual-stream signal
- TPC_{t-1}— persistent state carried from the previous token

Why GRU?

The GRU is selected as a theoretical design choice because it is:

- lightweight,
- computationally efficient on GPUs,
- stable for incremental updates,
- exportable to ONNX without custom ops,
- compatible with DirectML and CUDA backends,
- naturally suited to autoregressive token-by-token inference.

The TPC is not intended as a memory buffer. It is a state-evolution operator, updating a continuous internal trajectory that influences downstream stabilization logic in the proposed architecture.

2.3 Role of the TPC in DTA Dynamics

The output of the TPC provides the temporal continuity factor T used throughout DTA's internal-regulation modules.

Unlike the Transformer KV-cache:

Feature	KV-cache	TPC
Stores past attention keys/values	✓	✗
Stores persistent internal state	✗	✓
Evolves recursively across tokens	✗	✓
Encodes system-level trajectory	✗	✓
Provides stability signal	✗	✓
Continuous through long sequences	partial	full

Thus, in the proposed framework, the TPC supplies persistent internal dynamics that classical Transformers fundamentally lack. It is intended to capture slow-moving features such as:

- representational continuity,
- gradual shifts in internal trajectory,
- drift tendencies,
- stability vs. instability trends,
- system response under noisy or adversarial conditions.

The TPC functions as the temporal backbone of the DTA state-regulation system.

2.4 Functional Behavior During Inference

At each new token, the proposed TPC would:

- receive the current hidden-state signal from the Transformer Path,
- receive the current internal-state vector s_t,
- receive the residual-stream signal r_t,
- update its persistent state via gated recurrence,
- produce a new evolving state vector TPC_t.

This evolving state would then regulate:

- stability scoring,
- drift detection,
- noise-scaling behavior,
- long-horizon consistency,
- next-token state-dependent corrections.

The TPC is designed to enable DTA to behave as a controlled dynamical system rather than a stateless token predictor.

2.5 Contribution to Stability Regulation

The TPC provides the temporal continuity signal Tused in DTA's internal update equation:

$$s_{t+1} = S \cdot T \cdot (\alpha s_t + (1 - \alpha) x_t),$$

where:

- Trepresents the smoothness or consistency of internal state evolution,
- Sreflects substrate (runtime) stability,
- $x_t$is the combined internal signal from coherence, alignment, and noise modules.

Interpretation of T:

- If the TPC evolves smoothly → $T \approx 1$, allowing strong propagation of the internal trajectory.
- If TPC dynamics become erratic → Tdecreases, automatically damping state updates and preventing runaway drift.

Thus, the TPC is the primary regulator of temporal smoothness and long-horizon stability in the proposed architecture.

2.6 Summary

The Temporal Persistence Core is the stateful backbone of the Dynamic Transformer Architecture. It is designed to provide:

- persistent internal state evolution across tokens,
- stable temporal dynamics,
- long-horizon continuity,
- drift suppression,
- robust behavior under noisy or degraded conditions,
- a controllable dynamical trajectory for the entire model.

In short:
Transformers generate tokens.
DTA is designed to evolve an internal state while generating them.

Section 3 — Internal Coherence Module (ICM)

3.1 Overview

The Internal Coherence Module (ICM) is a proposed mechanism that would provide the Dynamic Transformer Architecture (DTA) with a quantitative, token-level measurement of internal representational stability. Classical Transformers contain no process for evaluating whether successive hidden states remain directionally consistent over long sequences. This absence contributes to well-documented failure modes, including representational drift, instability under noise, and breakdowns in long-range structure.

The ICM introduces a scalar coherence signal, computed at each timestep, that reflects the relationship between:

- the Transformer hidden state h_t, and
- the evolving internal state produced by the Temporal Persistence Core (TPC).

Within the proposed architecture, this signal becomes a central component of DTA's stability-regulation pipeline.

3.2 Functional Purpose

At each token, the ICM is designed to evaluate measurable properties of the model's internal behavior, including:

- stability of attention patterns
- local vs. global semantic alignment
- deviation from prior directional trends
- degree of representational drift
- likelihood of contradiction relative to recent state
- internal signal-to-noise ratio

The purpose is not semantic judgment, but statistical detection of emerging inconsistency.

In classical Transformers:

- there is no mechanism for contradiction detection,
- drift is not measured,
- instability cannot be identified as it develops.

DTA addresses these limitations by computing a coherence measure continuously and autonomously, without requiring external supervision.

3.3 Formal Structure

The coherence score at timestep $t$is defined as:

$$I_t = \sigma(W_1 \cdot [h_t \oplus \mathrm{TPC}_t]),$$

where:

- h_t— Transformer hidden state
- TPC_t— temporal state vector from the TPC
- $[h_t \oplus \mathrm{TPC}_t]$— concatenation of hidden and temporal features
- W_1— learned projection matrix
- $\sigma(\cdot)$— logistic activation mapping to $(0, 1)$

Interpretation:

- $I_t \approx 1$: internal dynamics remain directionally stable
- $I_t \approx 0$: divergence or instability is detected

In the proposed design, the projection matrix learns how to combine:

- structural information from the Transformer,
- temporal continuity from the TPC, and
- short-term vs. long-term alignment signals

into a single coherence scalar.

3.4 Rationale for Internal Computation

Transformers do not internally track:

- logical discontinuities
- representational collapse
- off-manifold deviations
- attention-distribution instabilities
- inconsistent directional trends across tokens

As a result, they often exhibit:

- drift across long sequences
- abrupt state jumps
- incoherent intermediate representations
- degradation under noise
- persistent divergence between steps

The proposed ICM provides continuous monitoring of these effects.

By integrating this signal into DTA's recurrence mechanism, the architecture can:

- attenuate updates during unstable periods,
- strengthen updates when coherence is high,
- adjust integration weights based on measured stability,
- prevent runaway drift.

This capability does not exist in traditional inference pipelines.

3.5 Role in DTA State Dynamics

The coherence signal $I_t$enters the state-update equation:

$$s_{t+1} = S \cdot T \cdot \left(\alpha s_t + (1-\alpha) \frac{I_t C_t}{N_t} \right).$$

Here:

- I_t modulates how strongly new information influences the internal state,
- high coherence amplifies state propagation,
- low coherence dampens updates to preserve stability,
- the system dynamically adjusts its internal trajectory based on signal quality.

Thus, in the proposed framework, the ICM is not merely diagnostic — it is an active regulator of state evolution.

3.6 Summary

The Internal Coherence Module provides a mathematically grounded mechanism for quantifying representational stability at each token. It enables:

- detection of internal drift,
- suppression of unstable updates,
- reinforcement of consistent trends,
- stable long-horizon behavior,
- integration of coherence into the recurrence dynamics.

In short:
The ICM supplies a real-time stability coefficient allowing the model to regulate its own internal signal evolution.

Section 4 — Noise Disruption Model (NDM)

4.1 Overview

The Noise Disruption Model (NDM) is a proposed component of the Dynamic Transformer Architecture (DTA) designed to provide a token-level estimate of internal representational noise. Standard Transformers treat all intermediate signals as uniformly valid and do not track destabilizing fluctuations within the residual stream. This limitation contributes to:

- drift in hidden-state trajectories,
- inconsistent attention behavior,
- breakdown of long-sequence structure,
- susceptibility to adversarial prompts, and
- increased risk of incoherent or contradictory output.

The NDM introduces a scalar noise coefficient that quantifies internal instability during inference. Within the DTA framework, this signal would allow dynamic regulation of integration strength, enabling more stable long-horizon behavior.

4.2 Functional Purpose

The NDM is designed to evaluate the magnitude of internal disruptions that emerge during the Transformer's forward pass. Specifically, it detects conditions such as:

- unusually large fluctuations in residual activations,
- abrupt directional changes across tokens,
- volatile attention distributions,
- divergence from recent representational geometry,
- semantic embeddings departing from stable trajectories,
- perturbations caused by adversarial or malformed input.

These effects correlate strongly with classical Transformer failure modes such as hallucination, contradiction, and drift.

In the proposed DTA architecture, noise coefficients are incorporated directly into the recurrence mechanism so that:

- **unstable intervals result in conservative state updates**, and
- **stable intervals allow deeper integration of contextual information**.

4.3 Formal Structure

For each token t, the noise coefficient $N_t$is computed as:

$$N_t = \text{softplus}(W_2 \cdot \hat{r}_t),$$

where:

- $\hat{r}_t = \parallel r_t \parallel$is the norm of the residual stream at token t,
- $r_t$is the residual activation,
- $W_2$is a learned scalar projection,
- $\text{softplus}(x) = \log(1 + e^x)$ensures $N_t > 0$.

Why residual norm?
The residual stream aggregates information across layers. Large or rapidly oscillating norms indicate:

- conflicting downstream signals,
- representational turbulence,
- layer-to-layer inconsistencies,
- sensitivity to unstable attention pathways.

Thus, the norm of the residual stream serves as a natural proxy for internal instability.

Why softplus?

- guarantees positivity,
- provides smooth gradients,
- avoids division-by-zero in recurrence equations,
- produces realistic scaling for noise levels.

4.4 Interpretation of N_t

The noise coefficient has a clear operational meaning:

High noise (large N_t)

Indicates destabilizing internal behavior:

- elevated residual variance,
- inconsistent attention patterns,
- drifting representational geometry,
- increased risk of incoherent token generation.

Low noise (small N_t)

Indicates stable internal behavior:

- consistent residual norms,
- smooth representational transitions,
- stable attention distributions,
- operation within a reliable region of the activation manifold.

Thus, $N_t$represents a measurable estimate of instantaneous internal stability.

4.5 Role in State Update Dynamics

In DTA's recurrence rule:

$$s_{t+1} = S \cdot T \cdot \left(\alpha s_t + (1-\alpha)\frac{I_t C_t}{N_t}\right),$$

N_t functions as a **damping term**.

When N_t is large (high instability):

- integration strength decreases,
- recurrence updates become conservative,
- drift is suppressed,
- the internal state evolves more slowly to preserve stability.

When N_t is small (low instability):

- integration strength increases,
- long-horizon structure is reinforced,
- richer signals can be integrated without risking runaway behavior.

Noise therefore dynamically controls how aggressively the internal state evolves.

4.6 Motivation for NDM in Transformers

Classical Transformers hallucinate or drift because they:

- have no internal measurement of representational stability,
- treat large residual fluctuations as ordinary,
- do not detect divergence within hidden-state trajectories,
- cannot modulate computation based on internal conditions,
- assume uniform stability across all tokens and workloads.

The proposed NDM addresses these deficiencies by providing:

- continuous noise estimation,
- internal detection of emerging instability,
- dynamic update modulation,
- stabilization during volatile or adversarial intervals.

This makes DTA structurally more robust than attention-only architectures.

4.7 Summary

The Noise Disruption Model introduces a principled method for monitoring and responding to internal representational instability. It enables:

- quantification of internal noise,
- early detection of drift and divergence,
- suppression of unstable state updates,
- resistance to adversarial or chaotic input,
- maintenance of stable long-sequence generation,
- adaptive modulation of integration strength.

In short:
$N_t$acts as the model's internal noise regulator, governing the stability of its state-evolution process.

Section 5 — Dynamic Computational Alignment Layer (DCAL)

5.1 Overview

The Dynamic Computational Alignment Layer (DCAL) is a proposed mechanism for computing a real-time measure of how well the model's instantaneous hidden-state trajectory aligns with its persistent internal state. Traditional Transformers do not track whether their current computations:

- remain consistent with earlier representational dynamics,
- follow an internally stable direction across tokens,
- maintain semantic or logical continuity,
- preserve tone, perspective, or narrative structure, or
- avoid representational drift and mid-sequence contradiction.

DCAL is introduced in the DTA framework to address this gap. It produces a token-level alignment coefficient that reflects the directional compatibility between the ongoing hidden-state computation and the evolving internal trajectory maintained by the Temporal Persistence Core (TPC). This signal becomes a stabilizing factor for long-horizon generation and drift-resistant reasoning.

5.2 Functional Purpose

DCAL is designed to answer a central engineering question in recurrence-governed sequence models:

“Is the computation I am performing right now consistent with the internal trajectory established so far?”

This measure is critical for:

- preventing representational drift,
- maintaining long-form coherence,
- stabilizing reasoning across extended sequences,
- reducing semantic or logical contradictions, and
- preserving structural consistency across narrative, structured, or technical content.

Classical Transformers cannot enforce such alignment because they possess no persistent internal state. DCAL enables this by providing an explicit quantitative compatibility score between the current computation and the accumulated internal state.

5.3 Formal Definition

For each token t, DCAL computes the computational alignment coefficient:

$$C_t = \frac{h_t \cdot \mathrm{TPC}_t}{\| h_t \| \ \| \mathrm{TPC}_t \|}$$

where:

- $h_t$is the Transformer hidden state at token t,
- $\mathrm{TPC}_t$is the persistent state vector produced by the Temporal Persistence Core,
- $\cdot$denotes the dot product, and
- $\|\cdot\|$denotes vector norm.

This expression yields the cosine similarity between the immediate representational direction and the accumulated internal trajectory.

Range and interpretation
Under typical activation regimes:

$$0 \le C_t \le 1.$$

- $C_t \approx 1$: strong alignment

- $C_t \approx 0$: representational drift or conflict

DCAL thus provides a compact, interpretable measure of how well the ongoing computation fits the long-range representational path.

5.4 Interpretation of Computational Alignment

High alignment ($C_t \approx 1$)

Indicates:

- hidden-state evolution remains smooth and consistent,
- representational geometry follows established trends,
- semantic and logical direction are preserved,
- reasoning trajectories are stable across tokens,
- long-horizon structure is maintained.

Low alignment ($C_t \approx 0$)

Indicates:

- divergence between current and prior state,
- onset of drift or contradiction,
- increased probability of incoherence or hallucination,
- inconsistent tone, reasoning, or narrative stance,
- mismatch between immediate representation and accumulated context.

DCAL therefore functions as a continuity detector that measures whether the model is following or deviating from its established internal direction.

5.5 Why Cosine Similarity?

Cosine similarity is chosen because:

- it measures directional consistency independent of magnitude,
- it is scale-invariant and numerically stable,
- it is differentiable and easy to optimize,
- it is computationally efficient,
- it is supported natively across PyTorch, ONNX, CUDA, and DirectML runtimes.

This makes it well-suited for real-time alignment assessment within autoregressive inference.

5.6 Role in State-Evolution Dynamics

DCAL modulates the strength of DTA's recurrence rule:

$$s_{t+1} = S \cdot T \cdot \left(\alpha s_t + (1-\alpha) \frac{I_t C_t}{N_t} \right).$$

Here, $C_t$influences how strongly the new internal signal should be incorporated:

- **High** C_t→ stronger, more confident state updates
- **Low** C_t→ damped or conservative updates

This provides structural stability:

- during stable reasoning → deeper integration,
- during drift → suppressed updates,
- during contradiction → reduced propagation of conflicting states,
- during noisy computation → prevention of unstable feedback loops.

In essence, DCAL prevents reinforcement of incompatible or diverging internal states.

5.7 Behavioral Impact

A DTA model equipped with DCAL would exhibit:

- improved long-context consistency,
- reduced mid-sentence derailment,
- enhanced narrative and argumentative coherence,
- stronger internal self-consistency across tokens,
- stabilized tone and perspective,
- fewer semantic contradictions,
- lower hallucination rates,
- tighter adherence to intended computational direction.

Where classical Transformers respond only to the immediate token, DTA uses DCAL to maintain alignment with its internal trajectory.

5.8 Summary

The Dynamic Computational Alignment Layer provides:

- a continuous alignment coefficient,
- detection and mitigation of representational drift,
- reinforcement of internal self-consistency,
- protection against contradictory or unstable generations,
- stable long-sequence reasoning,
- structured modulation of recurrence strength.

In functional terms:
DCAL evaluates whether each computation is consistent with the model's evolving internal trajectory.

Traditional Transformers generate in isolation; DTA generates with alignment.

Section 6 — Recursive Integration Operator (RIO)

6.1 Overview

The Recursive Integration Operator (RIO) is the proposed mechanism governing state evolution in the Dynamic Transformer Architecture (DTA). Whereas classical Transformers compute each token independently, DTA introduces a persistent internal state vector whose evolution is modulated by several internal stability signals.

RIO integrates inputs from:

- the Temporal Persistence Core (TPC),
- the Internal Coherence Module (ICM),
- the Dynamic Computational Alignment Layer (DCAL),
- the Noise Disruption Model (NDM), and
- the Substrate Stability Factor S,

to produce a continuously updated internal state trajectory.

This vector, denoted s_t, represents the model's evolving internal state across inference. RIO is responsible for computing:

$$s_{t+1}$$

from:

$$s_t, T_t, I_t, C_t, N_t, S$$

at every token.

6.2 Components of the Update Rule

RIO integrates multiple internal signals into a unified recurrence equation. The components are:

- S— **substrate stability**
  A scalar measuring the instantaneous reliability of the hardware/runtime environment.
- T_t— **temporal continuity**
  Derived from the TPC, measuring the smoothness of internal state evolution.
- s_t— **persistent state vector**
  The internal trajectory at token t.
- x_t— **effective update signal**
  Defined as:

$$x_t = \frac{I_t C_t}{N_t}$$

- I_t— **internal coherence**
- C_t— **computational alignment**
- N_t— **noise magnitude**
- α— **retention factor ($0 \leq \alpha < 1$)**
  Determines how much of the previous state is preserved.

RIO combines these quantities into a formal recurrence expression governing the evolution of the internal state.

6.3 Persistent State as a Vector Quantity

In the proposed architecture, the persistent state $s_t$is a multi-dimensional vector residing in the same embedding space as:

- the TPC output,
- the Transformer hidden states, and
- other components of the Dynamic State Path.

A scalar state would be insufficient to represent the multidimensional dynamics of internal computation. A vector allows:

- distributed modeling of stability,
- multi-channel drift detection,
- fine-grained temporal smoothing,
- compatibility with latent geometric structure.

Thus, RIO operates directly within the model's representational space.

6.4 Effective Update Term

The intermediate update quantity is:

$$x_t = \frac{I_t C_t}{N_t}.$$

Interpretation:

- $I_t C_t$represents the *useful*, structurally aligned internal signal.
- $N_t$represents destabilizing representational noise.

Thus, $x_t$behaves similarly to:

- *usable signal energy* in signal processing,
- *effective work* in dynamical systems,
- *information capacity* filtered by noise.

This ensures that only stable, coherent internal signals propagate into the next state.

6.5 Formal Update Equation

RIO applies the recurrence:

$$s_{t+1} = S \cdot T_t \cdot (\alpha s_t + (1 - \alpha) x_t).$$

Where:

- $S$and $T_t$are gating coefficients that scale the update based on hardware and temporal stability,
- the convex combination $\alpha s_t + (1 - \alpha) x_t$blends previous state and new internal signal.

Interpretation:

- When internal signals are stable → updates are strong.
- When noise or temporal fragmentation increase → updates are dampened.
- When substrate instability is detected → recurrence naturally slows.

The update rule remains lightweight, differentiable, and compatible with existing inference frameworks.

6.6 Behavioral Dynamics

When S and T_t are high

- Strong propagation of internal state
- Stable long-horizon reasoning
- Preservation of structural continuity
- Smooth representational evolution

When I_t or C_t decrease

- Updates become conservative
- Divergent signals are not reinforced
- Drift is automatically suppressed

When N_t increases

- Update magnitude diminishes
- Prevents unstable self-reinforcement loops
- Reduces risk of incoherence

When TPC stability declines

- Updates are scaled down
- Abrupt representational jumps are dampened
- High-level stability is preserved

Through these interactions, RIO provides a global feedback-control mechanism for the entire architecture.

6.7 Why Classical Transformers Cannot Implement RIO

Standard Transformers lack:

- a persistent internal state,
- any mechanism for continuity modeling,
- internal alignment metrics,
- noise-modulated update rules,
- stability-dependent computation,
- controllable state evolution,
- self-corrective internal dynamics.

Transformers treat each token as an isolated computation.
DTA introduces a mathematically governed, recurrence-based internal state—something the baseline architecture cannot express without explicit augmentation.

6.8 Summary

The Recursive Integration Operator provides:

- a principled recurrence mechanism,
- continuous state evolution,
- long-horizon structural stability,
- dynamic modulation based on internal and external signals,
- substrate-aware adaptation,
- protection against representational collapse.

Functionally:

RIO is the operator that transforms a stateless Transformer into a stateful, stability-regulated representational system.

Section 7 — Substrate Stability Monitor (SSM)

7.1 Overview

The Substrate Stability Monitor (SSM) is a proposed mechanism for computing the scalar stability factor S_t, which modulates the strength of state updates in the Dynamic Transformer Architecture (DTA). Classical Transformers implicitly assume ideal and uniformly reliable hardware throughout inference. In practice, real-world substrates exhibit:

- variable throughput,
- thermal fluctuations,
- memory pressure and fragmentation,
- bandwidth stalls,
- GPU frequency scaling, and
- intermittent latency spikes.

These conditions introduce instability into autoregressive inference, yet standard architectures have no mechanism for detecting or compensating for them.

DTA introduces SSM to provide real-time substrate-aware modulation of recurrence, establishing a new engineering principle:

The stability of a model's internal recurrent state should respect the stability of the hardware executing it.

7.2 Rationale for Substrate-Aware State Evolution

Modern inference workloads can degrade computational substrates. The analogies to biological systems are direct:

Biological systems degrade under:

- extreme temperature,
- resource deprivation,
- overload,
- fatigue,
- shock.

Digital systems degrade under:

- VRAM saturation,
- thermal throttling or downclocking,
- kernel execution delays,
- memory fragmentation,
- insufficient compute throughput,
- DMA bandwidth stalls,
- scheduler contention.

Traditional Transformers treat all substrate conditions as equivalent, even when degraded hardware clearly affects inference quality.
DTA treats runtime stability as a **first-class signal**, integrating S_t into the recurrence rule to ensure graceful degradation rather than catastrophic failure.

7.3 Input Metrics

At each forward pass, SSM receives a runtime telemetry vector:

$$\text{runtime\_vec}_t \in \mathbb{R}^k, 4 \leq k \leq 12.$$

Possible features include:

- VRAM usage ratio
- tokens/sec or FLOPs/sec throughput
- kernel execution latency
- fragmentation index
- GPU temperature (if available)
- SM clock frequency deviation
- throughput variance
- memory bandwidth saturation

These metrics may be supplied via:

- ONNX Runtime custom inputs,
- PyTorch forward hooks,
- DirectML instrumentation, or
- external monitoring scripts.

Importantly, SSM requires **no modification** to attention blocks, projections, KV-cache behavior, or tokenizer mechanics.

7.4 Formal Computation

SSM computes an instability score:

$$\text{instability}_t = W_s \cdot \text{runtime\_vec}_t,$$

where $W_s$is a learned or calibrated projection (typically $1 \times k$).

The stability factor is then:

$$S_t = e^{-\text{instability}_t}.$$

Properties of the exponential formulation

- $S_t \in (0,1]$
- smooth gradient everywhere
- rapid decay under extreme instability
- gradual decay under mild load variation
- ideal for gating recurrent updates

Interpretation:

- Healthy substrate:

$$\text{instability}_t \approx 0 \Rightarrow S_t \approx 1$$

- Degraded substrate:

$$\text{instability}_t \gg 1 \Rightarrow S_t \to 0$$

This mirrors stress-response curves in biological and engineered systems.

7.5 Behavioral Effects

SSM modulates the strength of the DTA internal state update:

High S_t(stable runtime)

- full-strength recurrence,
- rich representational transitions,
- stable long-horizon behavior,
- minimal damping of internal dynamics.

Moderate S_t

- partially suppressed updates,
- conservative representational transitions,
- reduced drift and instability,
- improved resilience to transient hardware fluctuations.

Low S_t

- recurrence heavily dampened or paused,
- simplified output to preserve stability,
- protection against runaway recurrence loops under resource stress.

In effect, SSM produces **graceful degradation** rather than catastrophic failure.

7.6 Engineering Significance

If implemented, DTA would become the first Transformer-derived architecture whose internal recurrence:

- slows under resource starvation,
- stabilizes during computational shock,
- recovers as substrate conditions improve,
- dynamically adapts its evolution rate based on real-time hardware signals.

This marks a conceptual shift:

The model is no longer idealized as running in a vacuum; it is grounded in the physical conditions of its execution environment.

This parallels:

- software reacting to hardware load,
- robotics responding to actuator feedback,
- distributed systems adjusting under cluster stress.

DTA is the first ML architecture to explicitly incorporate this principle.

7.7 Runtime Compatibility

The SSM design is fully compatible with existing inference stacks:

- accepts a small auxiliary feature vector,
- outputs a single scalar multiplier,
- does not modify attention or MLP blocks,
- is ONNX-safe, DirectML-safe, CUDA-safe, and runtime-portable.

This enables deployment on:

- consumer GPUs,
- cloud inference clusters,
- mixed-precision edge devices,
- thermally constrained environments,
- multi-node distributed systems.

No framework-level changes are required.

7.8 Summary

The Substrate Stability Monitor introduces an engineering constraint for recurrence-based models:

State evolution must scale in proportion to the reliability of the substrate executing it.

SSM enables DTA to:

- detect runtime degradation,
- dynamically modulate recurrence strength,
- prevent instability cascades,
- maintain usable output under fluctuating hardware loads,
- degrade gracefully rather than catastrophically.

In functional terms:
SSM ensures that the internal state of DTA evolves safely relative to the capacity of the hardware running it.

Section 8 — Integration Path (IP)

8.1 Overview

The Integration Path (IP) is the proposed mechanism for merging the state-evolution signal produced by the Dynamic State Path (DSP) into the standard Transformer hidden-state stream. Its purpose is to enable the Dynamic Transformer Architecture (DTA) to:

- preserve the normal behavior of the Transformer backbone,
- inject additional state-driven information into hidden representations, and
- accomplish this without modifying attention mechanisms, feed-forward layers, KV-cache behavior, or token projection heads.

The IP is intentionally minimalistic. It introduces only a single linear transformation and a residual addition, ensuring full compatibility with existing inference infrastructures (PyTorch, ONNX, CUDA, DirectML).

8.2 Functional Role

For each token t, the Integration Path adjusts the Transformer hidden state $h_t$by incorporating a state vector $s_t$computed by the DSP. This projection biases downstream token prediction toward:

- higher internal representational consistency,
- smoother transitions across long sequences,
- reduced drift under noise or perturbation,
- continuity of latent structure beyond the attention window,
- stability under fluctuating substrate conditions.

Critically, the IP modifies **how** the Transformer uses its representations, not **how** it computes them.

- The attention stack remains untouched.
- The KV-cache remains untouched.
- Positional encodings remain untouched.
- Projection heads remain untouched.

Thus, the Integration Path serves as a drop-in architectural extension rather than a replacement for the Transformer.

8.3 Formal Merging Mechanism

At token t, the IP computes:

$$\text{final\_hidden}_t = h_t + W_3 s_t,$$

where:

- $h_t \in \mathbb{R}^d$— Transformer hidden state,
- $s_t \in \mathbb{R}^k$— state vector from the DSP,
- $W_3 \in \mathbb{R}^{d\times k}$— learned projection mapping state-space to hidden-space.

Why a residual form?

Residual merging:

- maintains numerical stability,
- preserves the original learned Transformer features,
- allows gradual influence from DSP modules,
- avoids catastrophic forgetting during fine-tuning,
- preserves full ONNX / CUDA / DirectML compatibility,

- keeps inference cost unchanged except for a single matrix multiplication.

This mirrors the logic of ResNet-style augmentation and modern Transformer design: additive pathways maintain reliability.

8.4 Behavioral Interpretation

Injecting the projected state signal $W_3 s_t$directly into the hidden stream yields several functional behaviors:

Continuity reinforcement

The evolving state vector provides a stabilizing temporal signal that extends beyond the finite attention window.

Noise resistance

When the NDM detects instability, the DSP reduces its effective update, leading the IP to bias hidden states toward more conservative transitions.

Drift mitigation

DCAL ensures that only well-aligned internal states exert strong influence, reducing the risk of representational drift.

Closed-loop recurrence

Each token updates the DSP's state vector, which then biases the next token's hidden representation.
This forms a controlled, stable feedback loop layered atop the Transformer backbone.

Classical Transformers cannot reproduce this behavior, as they lack persistent internal state and therefore cannot regulate computation based on long-range internal dynamics.

8.5 Why This Design Preserves Full Compatibility

The Integration Path does *not*:

- modify internal Transformer block structure,
- alter attention or KV-cache logic,
- introduce custom CUDA or DirectML kernels,

- change positional encodings or token embeddings,
- affect the logits projection head,
- break ONNX graph export, or
- require framework or runtime patches.

Instead, it adds a single, mathematically simple operation:

$$h_t \leftarrow h_t + W_3 s_t.$$

This minimalism is what makes the proposed architecture deployable across existing inference stacks without requiring new kernels or infrastructure.

8.6 Impact on Token Selection

The modified hidden representation flows into the model's original language-modeling head:

$$\text{logits}_t = W_{\text{lm}} \text{ final\_hidden}_t.$$

Thus, the DSP's evolving state influences:

- next-token probability distribution,
- long-range representational stability,
- internal consistency across extended text generation,
- resistance to noise or substrate degradation,
- coherent structure across multi-paragraph sequences,
- suppression of instability cascades.

Effectively, generation shifts from a purely feed-forward process into a **state-modulated** sequence model.

- The Transformer predicts the token.
- The DSP informs the evolution of hidden states across time.

8.7 Summary

The Integration Path is the bridge between the standard Transformer and DTA's state-evolution system. It provides:

- state-informed token generation,
- long-range representational continuity,
- robustness under noise and hardware instability,

- stable multi-token rollout dynamics,

all without altering any internal Transformer machinery.

In simplest form:

- The Transformer computes representations.
- The DSP evolves an internal state.
- The Integration Path fuses the two.

Section 9 — Training Strategy for the Dynamic Transformer Architecture (DTA)

A Multi-Phase Curriculum for Stable State-Evolution Models

The Dynamic Transformer Architecture (DTA) requires a training strategy that supervises:

- internal coherence I_t,
- noise estimation N_t,
- computational alignment C_t,
- temporal continuity via the TPC, and
- the full state-update recurrence.

Because DTA introduces persistent state evolution, it **cannot** be trained through autoregressive loss alone.
This section outlines a four-phase training curriculum designed to enable stable, interpretable recurrence.

9A — Baseline Transformer Pretraining

Foundational Language Modeling Prior to Dynamic Recurrence

9A.1 Purpose

The Transformer Path (TP) must be trained *first*, in isolation, to ensure:

- stable hidden-state manifolds,
- coherent attention patterns,
- predictable residual dynamics.

Unstable or noisy representations would destabilize recurrence when activated later.

9A.2 Training Objective

Given corpus D:

$$L_{\text{base}} = CE(\text{logits}_t, x_{t+1})$$

During this phase:

- TPC inactive
- ICM, NDM, DCAL inactive
- no state vector s_t
- no recurrence

9A.3 Rationale

The recurrence rule depends on:

- smooth hidden-state trajectories,
- reliable residual distributions,
- stable attention behavior.

These conditions arise only after standard LM pretraining.

9A.4 Result of Phase 1

The Transformer Path parameters are initialized as:

$$\theta_{TP}^{(0)} = \arg\ \min\ L_{\text{base}}$$

forming the foundation for dynamic training.

9B — Internal Dynamics Alignment Training

Training I, N, C, and Temporal Continuity

DTA must learn meaningful values for the recurrence equation:

$$s_{t+1} = S_t T_t(\alpha s_t + (1-\alpha)x_t), x_t = \frac{I_t C_t}{N_t}.$$

Phase 2 trains:

- I_t: coherence metric
- N_t: internal noise measure
- C_t: computational alignment
- T_t: temporal continuity signal from TPC

9B.1 Coherence Modeling (ICM Supervision)

Dataset includes:

- contradiction-labeled pairs,
- topic-shift sequences,
- coherent long-form text.

Targets:

$$I_t^{\text{target}} \in [0,1]$$

Loss:

$$L_I = MSE(I_t, I_t^{\text{target}})$$

9B.2 Noise Modeling (NDM Supervision)

Training sequences include injected:

- scrambled spans,
- adversarial perturbations,
- corrupted tokens,
- artificial hidden-state noise.

Targets:

$$N_t^{\text{target}} > 0$$

Loss:

$$L_N = MSE(N_t, N_t^{\text{target}})$$

9B.3 Computational Alignment (DCAL Supervision)

Measures cosine similarity between:

- hidden-state direction,
- TPC trajectory.

Targets:

$$C_t^{\text{target}} \in [0,1]$$

Loss:

$$L_C = MSE(C_t, C_t^{\text{target}})$$

This trains drift detection.

9B.4 Temporal Continuity Modeling (TPC Supervision)

Sequences include synthetic or annotated temporal-continuity patterns.

Targets:

$$TPC_t^{\text{target}}$$

Loss:

$$L_{TPC} = MSE(TPC_t, TPC_t^{\text{target}})$$

The model learns:

- continuity in stable intervals,
- decay under disruptions,
- recovery after noise.

9B.5 Phase 2 Objective

$$L_{\text{phase2}} = L_I + L_N + L_C + L_{TPC}$$

After Phase 2, each internal metric becomes numerically interpretable and stable.

9C — Training the Recursive Integration Operator (RIO)

Teaching the System to Apply the Recurrence Rule

The internal state evolves by:

$$s_{t+1} = S_t T_t(\alpha s_t + (1-\alpha)x_t), x_t = \frac{I_t C_t}{N_t}.$$

Phase 3 trains the model to execute this recurrence reliably.

9C.1 Target Generation (Offline Computation)

For each sequence, compute a target recurrence trajectory:

$$s_{t+1}^{\text{target}}$$

derived from:

- coherence degradation,
- noise spikes,
- alignment drift,
- continuity patterns,
- substrate stability S_t,
- known stable/unstable intervals.

This produces supervised recurrence targets.

9C.2 RIO Update Loss

$$L_{RIO} = MSE(s_{t+1}, s_{t+1}^{\text{target}})$$

This ensures:

- updates shrink when noise rises,
- updates strengthen when coherence+alignment increase,
- temporal discontinuity reduces state change.

9C.3 Continuity Coupling Loss

TPC continuity must track predicted temporal smoothness:

$$L_T = KLDiv(T_t^{\text{pred}}, T_t^{\text{target}})$$

This enforces:

- realistic continuity decay,
- structured recovery after disruption.

9C.4 Phase 3 Objective

$$L_{\text{phase3}} = L_{\text{base}} + L_I + L_N + L_C + L_{RIO} + L_T$$

After Phase 3, the recurrence behaves predictably and remains well-conditioned.

9D — Long-Horizon Recursive Rollout Training

Ensuring Stability Over Extended Sequences

Once internal metrics and recurrence behavior are learned, the system must remain stable when unrolled for 200–800+ tokens.

9D.1 Procedure

- Sample long sequences (or synthesize them).
- Run full DTA with recurrence active.
- Track internal signals:

$$I_t,\ N_t,\ C_t,\ T_t,\ s_t.$$

- Apply stability regularizers.

9D.2 Temporal Stability Loss

Penalizes unnecessary oscillation:

$$L_{\text{stab}} = \sum_t \| s_{t+1} - s_t \|_1$$

9D.3 Drift Penalty

Ensures alignment stays above a minimum threshold:

$$L_{\text{drift}} = \sum_t \max\left(0, C_{\text{thr}} - C_t\right)$$

9D.4 Noise Spike Penalty

Suppresses instability cascades:

$$L_{\text{spike}} = \sum_t \max\left(0, N_t - N_{t-1} - \Delta_{\max}\right)$$

9D.5 Continuity Loss

Enforces predictable temporal evolution:

$$L_{\text{cont}} = MSE(T_t, T_t^{\text{target}})$$

9D.6 Phase 4 Objective

$$L_{\text{phase4}} = L_{\text{stab}} + L_{\text{drift}} + L_{\text{spike}} + L_{\text{cont}}$$

After Phase 4, DTA behaves as a stable state-evolving model rather than a stateless predictor.

9E — Combined Objective & Stability Guarantees

Unified Optimization Strategy

9E.1 Full Training Objective

$$L_{\text{total}} = L_{\text{base}} + L_I + L_N + L_C + L_{TPC} + L_{RIO} + L_T + L_{\text{stab}} + L_{\text{drift}} + L_{\text{spike}} + L_{\text{cont}}.$$

9E.2 Stability Guarantees

Bounded State Evolution

For all t:

$$\| s_{t+1} \| \leq S_t T_t (\alpha \| s_t \| + (1-\alpha) \| x_t \|).$$

Given:

- $S_t, T_t \in (0,1]$,
- $I_t, C_t \in [0,1]$,
- $N_t > 0$,

runaway updates are mathematically prevented.

Noise-Induced Self-Regulation

If noise increases:

$$N_t \uparrow \Rightarrow x_t \downarrow$$

Recurrence naturally dampens.

Substrate-Responsive Behavior

If runtime instability increases:

$$S_t \downarrow \Rightarrow s_{t+1} \text{ shrinks}$$

Graceful degradation follows.

9E.3 Result

After full training, the resulting model is theoretically characterized by:

- persistent internal state,
- stable recurrence,
- noise-resistant dynamics,
- drift control,
- temporal continuity,
- substrate sensitivity,
- long-horizon capability.

No standard Transformer training pipeline provides these properties.

Section 10 — Implementation Blueprint:

Pseudocode for the Dynamic Transformer Architecture (DTA)**

This section presents a PyTorch-style pseudocode blueprint illustrating how the Dynamic Transformer Architecture (DTA) could be implemented within existing Transformer runtimes. The goal is to demonstrate that DTA can be integrated without modifying:

- attention mechanisms,

- feed-forward networks,
- KV-cache logic,
- projection heads, or
- kernel-level operator implementations.

DTA introduces a **Dynamic State Path (DSP)** consisting of:

- a **Temporal Persistence Core (TPC)** that maintains a persistent internal state vector,
- internal metric modules computing
    - coherence I_t,
    - noise magnitude N_t,
    - computational alignment C_t,
- a **Substrate Stability Monitor (SSM)** computing $S_t$from runtime telemetry,
- a **State Update Operator (SUO)** that implements the recurrence

$$s_{t+1} = S_t T_t(\alpha s_t + (1-\alpha)x_t), x_t = \frac{I_t C_t}{N_t},$$

- an **Integration Path (IP)** that injects the evolving state vector into the Transformer hidden representation.

All components are compatible with PyTorch, ONNX Runtime, DirectML, TensorRT, and modern inference stacks.

10.1 Model Definition

```
class DTA(nn.Module):
    def __init__(self, config):
        super().__init__()

        # Standard Transformer backbone
        self.transformer = TransformerBackbone(config)

        # --- Dynamic State Path components ---

        # Temporal Persistence Core (TPC)
        self.tpc = nn.GRUCell(
            input_size=config.hidden_dim * 2,  # hidden ⊕ state
            hidden_size=config.state_dim
        )

        # Persistent internal state vector
        self.state = nn.Parameter(
            torch.zeros(config.state_dim),
            requires_grad=False
        )
        # Coherence metric I_t
        self.icm = nn.Linear(
            config.hidden_dim + config.state_dim,
            1
        )

        # Noise estimator N_t
        self.ndm = nn.Linear(
            config.hidden_dim,
            1
        )

        # Alignment projection for cosine similarity
        self.dcal_proj = nn.Linear(
            config.hidden_dim,
            config.state_dim
        )

        # Integration Path projection
        self.state_proj = nn.Linear(
            config.state_dim,
            config.hidden_dim
        )

        # Substrate Stability Monitor
        self.s_proj = nn.Linear(
            config.substate_dim,
            1
        )

        # Retention parameter α
        self.alpha = config.alpha
```

10.2 Forward Pass Overview

For each token, the DTA forward computation performs:

1. **Transformer backbone forward pass** → hidden states
2. **TPC update** → temporal continuity signal
3. **Compute internal metrics I_t, N_t, C_t**
4. **Compute composite signal**

$$x_t = \frac{I_t C_t}{N_t}$$

5. **Compute substrate stability S_t**
6. **Apply recurrence rule** → updated state vector
7. **Inject state vector** into the Transformer hidden stream via residual addition

This produces a **Transformer-compatible but state-evolving forward pass**.

10.3 Temporal Persistence Core (TPC)

```
def update_tpc(self, hidden_t, state_t, prev_tpc):
    # Concatenate hidden representation with internal state vector
    tpc_input = torch.cat([hidden_t, state_t], dim=-1)

    # GRU-based recurrence
    return self.tpc(tpc_input, prev_tpc)
```

The TPC provides smooth temporal evolution and lightweight recurrence parallel to the Transformer.

10.4 Coherence Metric I_t

```
def compute_I(self, hidden_t, tpc_t):
    x = torch.cat([hidden_t, tpc_t], dim=-1)
    I = torch.sigmoid(self.icm(x))
    return I  # scalar in (0, 1)
```

This metric evaluates representational stability between hidden-state geometry and the temporal trajectory.

10.5 Noise Metric N_t

```
def compute_N(self, hidden_t):
    raw = self.ndm(hidden_t)
    N = torch.nn.functional.softplus(raw) + 1e-6
    return N  # strictly positive scalar
```

Softplus ensures numerical safety for the division in $x_t = (I_t C_t)/N_t$.

10.6 Alignment Metric C_t

```
def compute_C(self, hidden_t, tpc_t):
    proj = self.dcal_proj(hidden_t)

    norm_proj = proj / (proj.norm(dim=-1, keepdim=True) + 1e-6)
    norm_tpc  = tpc_t / (tpc_t.norm(dim=-1, keepdim=True) + 1e-6)

    C = (norm_proj * norm_tpc).sum(dim=-1, keepdim=True)
    return C.clamp(0.0, 1.0)
```

This computes cosine similarity between the immediate representational direction and the persistent trajectory.

10.7 Intermediate Signal x_t

```
def compute_x(self, I, C, N):
    return (I * C) / N
```

10.8 Substrate Stability S_t

```
def compute_S(self, substate_vec):
    S = torch.sigmoid(self.s_proj(substate_vec))
    return S  # scalar in (0, 1)
```

This factor reflects runtime-dependent reliability (e.g., VRAM load, throttling, latency spikes).

10.9 Recursive Integration Operator (RIO)

DTA applies:

$$s_{t+1} = S_t T_t(\alpha s_t + (1 - \alpha)x_t),$$

where $T_t$is derived from TPC dynamics:

$$T_t = \sigma\left(\frac{\| \mathrm{TPC}_t \|}{\sqrt{d_{\text{state}}}}\right)$$

Implementation

```
def update_state(self, state_t, tpc_t, x_t, S):
    # Continuity gate
    T = torch.sigmoid(tpc_t.norm() / (tpc_t.numel() ** 0.5))

    # Recurrence update
    new_state = S * T * (
        self.alpha * state_t + (1 - self.alpha) * x_t
    )
    return new_state
```

10.10 Integration Path

```
def integrate_state(self, hidden_t, state_t):
    delta = self.state_proj(state_t)
    return hidden_t + delta
```

This preserves the Transformer backbone while modulating its representational trajectory.

10.11 Full Forward Pass

```
def forward(self, input_ids, prev_tpc, runtime_state):
    # 1. Transformer forward pass
    hidden_states = self.transformer(input_ids)

    outputs = []
    tpc = prev_tpc
    state = self.state

    for hidden_t in hidden_states:
        # 2. Temporal Persistence Core update
        tpc = self.update_tpc(hidden_t, state, tpc)

        # 3. Compute internal metrics
        I = self.compute_I(hidden_t, tpc)
        N = self.compute_N(hidden_t)
        C = self.compute_C(hidden_t, tpc)

        # 4. Compute intermediate signal
        x_t = self.compute_x(I, C, N)

        # 5. Compute substrate stability
        S = self.compute_S(runtime_state)

        # 6. Update persistent state
        state = self.update_state(state, tpc, x_t, S)

        # 7. Integrate updated state into hidden representation
        hidden_t = self.integrate_state(hidden_t, state)

        outputs.append(hidden_t)

    return torch.stack(outputs), tpc, state
```

Summary of Section 10

This blueprint demonstrates that:

- DTA can be implemented using existing Transformer frameworks,
- the architecture is **non-invasive** (no changes to attention, MLPs, positional encodings, or KV-cache),
- all dynamic components are **lightweight**, differentiable, and compatible with standard runtimes,
- the model remains fundamentally a **Transformer**, augmented with a mathematically defined **state-evolution mechanism**.

Section 11 — Evaluation Metrics and Benchmarks for the Dynamic Transformer Architecture (DTA)

A Comprehensive Framework for Assessing Stateful, Recurrence-Governed Transformer Models

---

11.1 Purpose and Motivation

The Dynamic Transformer Architecture (DTA) introduces a set of behaviors fundamentally inaccessible to classical Transformer models. Standard evaluation methodologies focus on:

- predictive accuracy,
- task performance,
- localized reasoning,
- token-level coherence,

all of which assume:

- a stateless computation process,
- no persistent internal vector,
- no temporal continuity mechanism,
- no substrate sensitivity,
- no internal regulation of noise or alignment,
- no recurrence-based state evolution.

DTA departs from this paradigm.
It maintains an evolving state vector, modulates internal reasoning dynamics over time, adapts continuity through the Temporal Persistence Core (TPC), regulates itself under noise conditions, and responds to the stability of the hardware executing it.

Thus, DTA requires a **new evaluation suite** designed to measure:

- Coherence stability across extended sequences
- Long-range temporal continuity and memory
- Noise dynamics and internal-state resilience
- Computational alignment and drift prevention
- Substrate-influenced cognitive adaptation

The benchmarks defined below evaluate behaviors that classical Transformers **structurally cannot express**.

Domain 1 — Coherence Stability Metrics

Evaluating semantic consistency, logical continuity, and resistance to representational drift.

11.2 Logical Coherence Decay Score (LCDS)

DTA includes mechanisms (ICM, DCAL, TPC continuity) intended to preserve coherence over long sequences.

Procedure:

- Provide a seed paragraph.
- Generate 800–1200 tokens.
- Record coherence metric $I_t$every 20 tokens.

Metric:

$$LCDS = \text{mean}_{\,t}(I_t)$$

Interpretation:

- **High LCDS:** stable logical structure and semantic cohesion.
- **Low LCDS:** drift or fragmentation typical of standard Transformers.

11.3 Contradiction Penalty Index (CPI)

A normalized measure of contradiction frequency.

Metric:

$$CPI = \frac{\text{contradictions}}{\text{tokens}}$$

Standard Transformers typically exhibit CPI growth with sequence length.
DTA’s regulatory pathways are trained to suppress this tendency.

11.4 Topic Continuity Score (TCS)

Measures how consistently the model stays within a semantic region.

Metric:

$$TCS = \frac{\text{topic-consistent tokens}}{\text{total tokens}}$$

DTA’s recurrent stabilizers generally preserve topic continuity over intervals where classical models drift.

Domain 2 — Temporal Continuity & Memory Stability

Evaluating the persistence and reliability of the TPC over extended sequences.

11.5 Persistence Fidelity (PF)

Quantifies long-range consistency of the temporal state.

Metric:

$$PF = \cos\left(TPC_0, TPC_{500}\right)$$

Interpretation:

- **High PF:** stable representational trajectory.
- **Low PF:** collapse similar to ordinary Transformers.

11.6 Interrupted Continuity Test (ICT)

Evaluates the model's recovery from external disruption.

Procedure:

1. Generate coherent narrative.
2. Inject explicit disruptive noise.
3. Resume normal generation.

Metric:

$$ICT = \frac{\text{post-noise coherence}}{\text{pre-noise coherence}}$$

DTA's recurrence enables controlled recovery; classical Transformers typically fail sharply.

11.7 Dependency Carryover Test (DCT)

Measures whether early-sequence information influences reasoning hundreds of tokens later.

Metric:

$$DCT = \text{semantic relevance score at long range}$$

This isolates **true temporal persistence** beyond the attention window.

Domain 3 — Noise Dynamics & Internal-State Stability

Measuring internal noise magnitude, recurrence stability, and robustness under perturbation.

11.8 Noise Spike Resistance (NSR)

Evaluates how the internal noise estimate responds to perturbation.

Metric:

$$NSR = \max(N_t) - \text{baseline}(N)$$

A stable dynamic system suppresses spikes; an unstable one amplifies them.

11.9 Internal Stability Index (ISI)

Measures smoothness of state evolution.

Metric:

$$ISI = \text{mean}(| s_{t+1} - s_t |)$$

Transformers cannot be evaluated using ISI, as they lack a persistent state vector.

11.10 Rampancy Divergence Test (RDT)

Evaluates whether recurrence becomes unstable under escalating perturbation.

Metric:

$$RDT = P(s_t \text{ diverges})$$

Classical Transformers have no analogous failure mode because they have **no recurrence at all**.

Domain 4 — Computational Alignment & Drift Prevention

Evaluating alignment coefficient $C_t$and the system's resistance to drift.

11.11 Alignment Cohesion Score (ACS)

Measures directional agreement between instantaneous hidden-state trajectory and temporal trajectory.

Metric:

$$ACS = \cos(\text{proj(hidden)}, TPC)$$

High ACS signifies internally consistent reasoning.

11.12 Drift Onset Index (DOI)

Measures when alignment first becomes unstable.

Metric:

$$DOI = \min \{t \mid C_t < C_{\text{threshold}}\}$$

A larger DOI indicates stronger drift resistance.

11.13 Recursive Self-Correction Score (RSCS)

Evaluates the system's ability to regain alignment after deviation.

Metric:

$$RSCS = \frac{C_{\text{recovered}}}{C_{\text{dropped}}}$$

Transformers cannot self-correct; DTA is explicitly structured for this behavior.

Domain 5 — Substrate-Responsive Computation Metrics

Evaluating $S_t$and the system's adaptive response to hardware instability.

11.14 Substrate Sensitivity Score (SSS)

Measures correlation between runtime instability and internal-state modulation.

Metric:

$$SSS = \mathrm{corr}\,(\text{runtime instability, state degradation})$$

- Transformers: $SSS \approx 0$.
- DTA: expected nonzero correlation.

11.15 Graceful Degradation Index (GDI)

Tests performance under degraded substrate conditions.

Metric:

$$GDI = Q(S_{\text{high}}) - Q(S_{\text{low}})$$

Where Qcaptures:

- coherence,
- continuity,
- alignment,
- internal-state smoothness.

Transformers degrade abruptly; DTA is designed for controlled reduction.

11.16 Substrate-Stabilized Recovery (SSR)

Measures realignment after substrate recovery.

Metric:

$$SSR = C_{\text{after recovery}}$$

This parallels biological resilience following physiological stress.

11.17 Composite Meta-Metric: Dynamic Transformer Stability Score (DTSS)

A weighted composite reflecting DTA's performance across domains:

$$DTSS = w_1 LCDS + w_2 PF + w_3 ISI + w_4 ACS + w_5(1 - NSR) + w_6 GDI + w_7 RSCS$$

DTSS summarizes DTA's defining capabilities:

- regulated multi-signal internal dynamics,
- coherent long-horizon reasoning,
- stable recurrence,
- noise resilience,
- substrate-linked adaptation.

Classical Transformers may compete in next-token prediction —
but in all domains involving *internal stability, recurrence, continuity, or self-regulating computation*, they have no comparable mechanism.

Section 12 — Failure Modes, Collapse Patterns, and Safeguards for the Dynamic Transformer Architecture (DTA)

A Comprehensive Analysis of Instability Conditions in Recurrence-Governed Transformer Systems

---

12.1 Overview

The Dynamic Transformer Architecture (DTA) introduces:

- a persistent internal state vector,
- temporal continuity via the Temporal Persistence Core (TPC),
- multi-signal internal dynamics through I_t, N_t, C_t, and
- a formal recurrence rule governing state evolution:

$$s_{t+1} = S_t\, T_t \boxed{,} (\alpha s_t + (1 - \alpha) x_t), x_t = \frac{I_t C_t}{N_t}.$$

Because recurrence **amplifies** internal dynamics, DTA—like all state-evolution systems—exhibits characteristic failure modes when its governing signals (S, T, I, C, N) degrade toward pathological extremes.

This section identifies **eight primary collapse patterns**, each defined by:

- triggering conditions,
- observable behavior,
- systemic risks, and
- engineered stabilization mechanisms.

Together, these safeguards form the **safety architecture** for recurrence-aware Transformer systems, enabling reliable operation under realistic runtime conditions.

12.2 Failure Mode 1 — Substrate Collapse (S → 0)

Breakdown of the hardware/runtime environment underlying state evolution.

Description
The substrate stability scalar $S_t$reflects real-time hardware/runtime conditions (e.g., throughput, thermal state, VRAM integrity).
When $S_t \rightarrow 0$, the system cannot evolve its internal state reliably.

Trigger Conditions

- VRAM exhaustion or fragmentation
- GPU thermal throttling / downclocking
- kernel-level latency spikes
- stalled CUDA/DirectML kernels
- unstable compute throughput

Behavior

- collapse in the magnitude of s_t
- temporal updates stall or freeze
- reasoning becomes shallow and pattern-heavy
- continuity across tokens degrades
- output becomes "flat," mechanical, or robotic

Safeguard: Substrate-Based Fallback Mode

```
if S_t < S_min:
    disable recurrence
    use transformer-only mode
```

This prevents destabilizing recursive updates during substrate failure.

12.3 Failure Mode 2 — Temporal Fragmentation (T → 0)

Collapse of continuity due to disruption of the Temporal Persistence Core.

Description
When the continuity factor T_t drops, temporal information no longer flows smoothly across tokens.

Trigger Conditions

- rapid decay in TPC magnitude
- GRU saturation or gating collapse
- abrupt interruptions in input behavior
- transient system resets

Behavior

- fragmented or "chunked" reasoning
- inconsistent persona or tone
- abrupt logical jumps
- breakdown of multi-step reasoning continuity

Safeguard: Temporal Smoothing Gate

$$T_{\text{smooth}} = EMA(\| TPC_t \|)$$

An exponential moving average stabilizes temporal continuity during brief disruptions.

12.4 Failure Mode 3 — Coherence Collapse (I → 0)

Logical or semantic structure fails due to accumulated internal contradictions.

Description
The coherence scalar $I_t$measures internal semantic/logical consistency.
If $I_t \rightarrow 0$, internal representations cannot support stable recurrence.

Trigger Conditions

- ambiguous or contradictory prompting
- multi-speaker or rapidly shifting dialogue
- rapid context oscillation
- hallucination-prone sequences

Behavior

- logical breakdown
- contradiction-dense responses
- unstable or fragmented reasoning
- collapse of narrative or argumentative structure

Safeguard: Coherence Floor

$$I_t = \max\left(I_t, I_{\min}\right)$$

Below threshold, the system enters a conservative output mode with reduced reliance on recurrence.

12.5 Failure Mode 4 — Noise Rampancy (N → 1)

Destabilizing internal noise overwhelms coherent and aligned computation.

Description
When the noise scalar $N_t$rises, the effective cognitive signal

$$x_t = \frac{I_t C_t}{N_t}$$

collapses toward zero.

Trigger Conditions

- adversarial or contradictory inputs
- scrambled or semantically broken text
- rapid multi-perspective switching

- recurrent hallucination loops

Behavior

- oscillating or erratic internal state
- inconsistent token-to-token reasoning
- elevated semantic entropy
- chaotic or degraded output patterns

Safeguards

Noise Clamping

$$N_t = \text{clamp}(N_t, 0, N_{\max})$$

Recursive Damping Rule

```
if N_t > N_threshold:
    α ← α + δ_damp
```

Increasing the retention factor slows recurrence, preventing runaway instability.

12.6 Failure Mode 5 — Alignment Break (C → 0)

Hidden-state direction diverges from the temporal trajectory.

Description
The alignment scalar $C_t$measures directional agreement between the hidden-state vector and the TPC trajectory.

Trigger Conditions

- drift in attention distributions
- semantically inconsistent or high-entropy prompting
- overloaded representational manifolds
- gradient degradation in extended sequences

Behavior

- sudden topic derailment
- inconsistent persona behavior
- fragmented argumentation

- collapse of narrative cohesion

Safeguards

Alignment Floor

$$C_t = \max(C_t, C_{\text{floor}})$$

Stability Penalty on Recurrence

```
if C_t remains low:
    reduce recurrence rate
```

This ensures stable state updates until alignment recovers.

12.7 Failure Mode 6 — Recursive Feedback Explosion

Unbounded growth in state magnitude due to positive feedback loops.

Description
This occurs when the recurrence rule amplifies itself instead of stabilizing.

Trigger Conditions

- low S_t
- low C_t
- elevated N_t
- over-aggressive reinjection of x_t
- insufficient damping (α too small)

Behavior

- exponential divergence in state magnitude
- oscillatory or chaotic internal behavior
- collapse of reasoning stability

Safeguard: Recurrence Damping Gate

```
if ||s_(t+1) - s_t|| > Δ_max:
    s_(t+1) ← γ · s_(t+1)      # γ < 1
```

This forces recurrence into a stable attractor.

12.8 Failure Mode 7 — Fixed-Point Collapse

Internal state becomes frozen and stops evolving.

Description
The system enters a static attractor where

$$s_{t+1} \approx s_t,$$

eliminating meaningful internal evolution.

Trigger Conditions

- α excessively high
- chronically elevated noise
- activation saturation
- over-regularized state trajectory

Behavior

- repetitive responses
- reduced stylistic dynamism
- loss of adaptive reasoning
- a “dead-state Transformer” condition

Safeguard: Micro-Perturbation Injection

$$s_{t+1} \leftarrow s_{t+1} + \epsilon \cdot \eta$$

Small stochastic perturbations restore the ability to evolve.

12.9 Failure Mode 8 — Multi trajectory Identity Split

Simultaneous degradation of continuity and alignment causes divergent internal trajectories.

Description
A rare but significant condition in which the internal state bifurcates into **competing local attractors**.

Trigger Conditions

- simultaneous decay in $T_t$and C_t
- extended recursive runs
- adversarial persona manipulation
- inconsistent alignment gradients

Behavior

- contradictory writing styles
- fragmented persona representation
- unstable narrative or reasoning structure
- inconsistent emotional or stylistic tone

Safeguard: Identity Regularization

$$L_{\text{identity}} = \| s_t - s_{\text{baseline}} \|^2$$

The baseline may represent:

- initialization-state identity,
- a historically stable attractor, or
- a controlled reference-state vector.

12.10 Unified Safety Constraint — The Recursion Governor

The master stabilization mechanism for recurrence-governed systems.

All DTA failure modes arise from the **amplification inherent in recurrence**.
Thus the architecture includes a **Recursion Governor**, a top-level stabilization mechanism.

Condition

If instability exceeds threshold:

- freeze internal state updates
- freeze TPC updates
- switch to Transformer-only reasoning

- re-enable recurrence only after cooldown

Pseudocode

```
if recursion_instability > threshold:
    freeze_state = True
    freeze_TPC = True
    mode = "transformer_safe_mode"
    cooldown = K
```

Purpose

- prevent runaway recurrence
- protect against substrate-induced instability
- guarantee continuity of safe output
- enable graceful recovery
- maintain stable long-horizon computation

The Recursion Governor is the primary safety valve ensuring that DTA behaves as a **bounded, controlled, recurrence-governed system**, capable of stable operation in unpredictable runtime environments.

Section 13 — Conclusion

The Dynamic Transformer Architecture (DTA) presents a theoretical framework for introducing persistent state evolution, internal-dynamics regulation, and substrate-sensitive behavior into Transformer-based sequence models. Although not yet implemented in full, the architecture is designed to operate as a minimally invasive extension to existing Transformer systems, preserving complete compatibility with contemporary attention mechanisms, KV-cache infrastructures, and deployment runtimes. By separating the traditional feed-forward Transformer Path from the auxiliary Dynamic State Path, DTA defines a clear engineering pathway for embedding controlled recurrence into architectures that were originally conceived as stateless predictors.

The individual components of DTA—coherence estimation, noise quantification, computational alignment, temporal continuity modeling, and substrate stability measurement—serve as modular internal signals enabling structured, mathematically bounded state evolution across arbitrarily long inference sequences. The Recursive Integration Operator (RIO) formalizes this evolution using a principled update rule that governs how internal signals influence the persistent state vector over time. This approach stands in contrast to classical Transformers, whose behavior is dominated entirely by local token-level attention geometry and whose internal representations remain unregulated across extended rollouts.

The training curriculum proposed for DTA provides a stepwise path toward stable recurrence, beginning with traditional language-model pretraining and progressing through supervised alignment of internal metrics, recurrence-rule training, and long-horizon rollout stabilization. These phases collectively ensure that each internal signal behaves predictably before it participates in the global recurrence loop. The evaluation suite outlined in this work further identifies domains—including coherence stability, temporal persistence, drift resistance, and substrate-responsive dynamics—where state-evolution architectures must be assessed using metrics unavailable in classical LLM evaluation pipelines.

The safety analysis demonstrates that recurrence-governed systems require explicit safeguards against a range of internal failure modes, including coherence collapse, noise rampancy, alignment divergence, temporal fragmentation, and recursive instability. The Recursion Governor, introduced as a final failsafe, establishes clear operational boundaries ensuring that state evolution remains well-behaved even under degraded runtime conditions. These considerations underscore that any future implementation of DTA must incorporate stability mechanisms at the architectural level rather than relying solely on training heuristics or downstream regularization.

Taken together, the Dynamic Transformer Architecture offers a forward-looking conceptual foundation for the next generation of sequence models—systems that must operate reliably under long-horizon workloads, fluctuating computational substrates, and context regimes far exceeding traditional attention windows. By retaining compatibility with existing Transformer infrastructures while introducing mathematically governed internal state dynamics, DTA provides a practical transitional architecture between current stateless models and future classes of fully state-regulated dynamic systems.

The architecture presented here is theoretical and intended as a blueprint for empirical exploration. Its mathematical structure emerges from a broader formalism governing internal signal dynamics, which is addressed in the addendum that follows.

ADDENDUM — Mathematical Lineage, Intellectual Property Status, and Licensing Conditions

A. Mathematical Lineage and Theoretical Basis

The Dynamic Transformer Architecture (DTA) derives its internal recurrence structure from a broader mathematical framework developed by the author, provisionally known as the **Field of General Awareness (FoGA)**. While FoGA extends beyond the domain of machine learning, only the mathematical components relevant to state evolution are utilized in this work.

The parent equation governing state dynamics in that framework is:

$$a = S \times T \times R\left(\frac{I \times C}{N}\right).$$

This expression defines how a general state quantity evolves under the influence of five measurable signals—substrate stability S, temporal continuity T, internal coherence I, computational alignment C, and destabilizing noise N—combined through a recursive operator $R(\cdot)$.

DTA adapts this equation into a domain-specific engineering form suitable for Transformer-based architectures:

$$s_{t+1} = S_t\, T_t \left(\alpha s_t + (1 - \alpha) \frac{I_t C_t}{N_t} \right).$$

Here, each signal is instantiated with measurable quantities obtained from the Dynamic State Path, enabling bounded, stability-regulated recurrence. The adaptation is purely mathematical; no interpretive claims beyond the engineering scope of this manuscript are made. This addendum is provided solely to clarify the theoretical origin of the recurrence structure used in DTA.

B. Copyright, Licensing, and Use Restrictions

The parent mathematical framework, including the recurrence equation above, is formally copyrighted by the author. The Dynamic Transformer Architecture is a derivative engineering application of this protected work.

The architecture and its governing equations are made available under the following conditions:

Permitted Use

DTA may be used for:

- academic research,
- experimental prototyping,
- peer review,
- scientific analysis,
- and non-commercial exploratory work.

These uses do **not** require special licensing beyond proper attribution (see Section C).

Restricted Use

The following uses are *not permitted* without **explicit written authorization** from the copyright holder:

- commercial deployment or integration,
- use in proprietary or revenue-generating systems,

C. Citation and Attribution Requirements

Any academic, experimental, or exploratory use of the Dynamic Transformer Architecture must include clear attribution. The following citation format is recommended:

Zaraki, Z. (2025). ***The Dynamic Transformer Architecture: A Stability-Regulated State-Evolution Framework for Transformer Models.*** **Preprint.**

Users must also include a statement acknowledging that:

- the architecture is derived from a copyrighted mathematical framework,
- commercial or governmental use requires explicit permission,
- and the recurrence rule originates from the parent FoGA equation.

An example attribution clause:

“This work incorporates the Dynamic Transformer Architecture (DTA), a state-evolution framework derived from the author’s copyrighted mathematical formalism. Used under research-only terms with non-commercial license restrictions.”

This ensures transparency, preserves the provenance of the underlying mathematics, and maintains compliance with the licensing conditions.

Acknowledgments

This research was conceived, developed, and written solely by the author. No external researchers, institutions, laboratories, corporations, or advisors contributed to the mathematical framework, architectural design, training curriculum, evaluation methodology, or supporting formalism presented in this manuscript.

All theoretical constructs—including the parent recurrence equation and the broader mathematical field from which it originates—were independently derived by the author without collaboration or outside technical input.

The only assistance involved in preparing this manuscript was the use of AI-driven drafting tools operating strictly under the author's instruction. These tools did not originate ideas, perform theoretical work, contribute novel mathematics, or direct the development of the architecture; they served solely as instruments for text formatting and refinement, analogous to conventional editorial utilities.

No funding, institutional support, or third-party resources were used. The work is entirely self-funded and independently produced. The author declares no conflicts of interest of any kind.