A Constraint-First Architecture for Generative AI: From Token Prediction to Hierarchical Locking

Author: Zhe Gao

Affiliation: Independent Researcher

Date: July 2026

Abstract

Current large language models (LLMs) generate text through autoregressive token prediction—a fundamentally bottom-up process in which the model samples from a probability distribution over the next token, relying on statistical patterns learned from training data. While this approach has achieved remarkable success, it suffers from a structural limitation: constraints (safety, factual accuracy, logical consistency, stylistic coherence) are applied after generation, through post-hoc filtering, RLHF, or inference-time prompting. This is not a matter of "stronger constraints"—it is a matter of when and where constraints are applied.

We propose a fundamentally different architecture. Drawing on the "1=N" hierarchical framework, we invert the generation direction: the top-level semantic goal (the "1") defines the legitimate generation space a priori, and generation proceeds within this constrained space. A dynamic Constraint Strength parameter (λ) and a real-time Feedback Signal (δ) form a closed-loop control system that continuously monitors and adjusts the generation process. When novel, effective patterns emerge during generation, they are consolidated into new constraint layers—creating a recursive structure in which each generation step both produces output and updates the rules that govern subsequent output.

This paper presents the architectural blueprint: the conceptual foundation, the core mechanisms, the formal definitions, and a preliminary implementation framework. We position this work against existing approaches (RLHF, Constitutional AI, constrained decoding) and outline a pathway toward empirical validation.

Keywords: Constraint-First Generation; Hierarchical Locking; Emergence Consolidation; Generative AI Architecture; 1=N Principle

---

1. Introduction: A Structural Problem in Current Generative AI

The dominant paradigm in generative AI is, at its core, a bottom-up process. Given a prompt, the model generates a response token by token, each token sampled from a probability distribution conditioned on all previous tokens. This is a statistical approximation of "what comes next" learned from vast amounts of human text.

This paradigm has produced remarkable results. It has given us models that can write poetry, solve math problems, and engage in philosophical dialogue. But it has also produced a persistent set of problems:

•**Hallucinations:** The model generates content that is fluent but factually false.
•**Logical inconsistency:** Long-form generations often contradict themselves.
•**Boundary violations:** The model produces content that violates safety, ethical, or task-specific constraints.
•**Opacity:** We cannot trace why a particular output was generated.

The standard response to these problems has been post-hoc constraint application: RLHF (reinforcement learning from human feedback) modifies the model's behavior after training; Constitutional AI imposes constraints during fine-tuning; inference-time prompting instructs the model to "be careful" or "stay within bounds"; and external filters screen outputs after generation.

We argue that this is a structural limitation, not a matter of insufficient constraint strength. The current architecture applies constraints at the wrong point in the generation process. Constraints are applied after the model has already explored the space of possibilities—when it is too late to prevent the model from generating an invalid output, only to filter or correct it.

The central thesis of this paper is: To solve these problems at their root, we must invert the generation direction. Constraints should be applied before and throughout generation, not after it. The model should generate within a pre-defined legitimate space, rather than generating freely and then being filtered.

It is worth clarifying what this proposal does **not** entail. We are not advocating for the elimination of generative diversity, nor for imposing a static rule cage that would reduce the model to a deterministic template filler. Rather, we draw a fundamental distinction between two kinds of variation:

•**Productive creativity:** variation that operates within the task-defined legitimate space, exploring alternative phrasings, argument structures, and stylistic expressions that serve the generative goal.
•**Unproductive deviation:** variation that exits the legitimate space, producing hallucinations, logical contradictions, factual errors, or boundary violations that undermine the output's utility and reliability.

The architecture preserves—and in fact encourages—the former through its dynamic constraint mechanism: when the output remains within bounds ($\delta \approx 0$), constraint strength λ relaxes, granting the generator ample freedom to explore diverse expressions within the task space. It structurally precludes the latter by defining the generation space a priori and enforcing its boundaries in real time. In this sense, the framework is not anti-creative; it is **anti-hallucination by design**, and it treats the so-called "creativity" that produces falsehoods as a structural defect to be eliminated, not a feature to be preserved.

---

2. Conceptual Foundation: The "1=N" Framework

This architecture is grounded in the "1=N" hierarchical framework (Gao, 2025, 2026), which provides a formal ontology of functional hierarchy.

2.1 Core Definitions

Term	Symbol	Definition
Functional Hierarchy	L	An organizational level at which a system stably exercises a unique, irreducible function
Whole	1	At a given level L, the system that can be identified as an independent unit exercising the core function of that level
Parts	N	The set of components at the lower level (L-1) that constitute the whole "1"

Term	Symbol	Definition
Constraint Relation	R	The specific interaction pattern that connects the N parts, enabling them to collectively constitute the higher-level whole "1"
Constraint Strength	$\lambda \in [0,1]$	The degree to which the top-level constraint is enforced on the lower level
Feedback Signal	$\delta \geq 0$	A signal generated when the lower-level state exceeds the top-level constraint boundary; intensity proportional to the degree of overload

2.2 Two Foundational Theses

Thesis 1: The identity of a system derives from the stacking of impossibilities, not the proliferation of possibilities.

A system is this system rather than another not because of what it can produce, but because of what it has rejected. Constraints are not external limitations—they are the structural basis of system identity.

Thesis 2: Emergence is temporary; stability requires locking.

Emergence—the appearance of new structures or patterns from lower-level interactions—is real, but it is not the end of the story. An emergent pattern has no stability unless it is subsequently locked—i.e., consolidated into a new constraint layer that governs future behavior. In this framework, emergence and constraint are not opposites; they are two phases of a single cycle: **Emergence → Locking → New Constraints → Further Emergence**.

2.3 Implications for Generative AI

If we treat a generative model as a hierarchical system:

•The **top-level "1"** is the semantic goal of the generation task (e.g., "write a factual summary of topic X" or "generate a grammatically correct and logically coherent argument").
•The **lower-level "N"** is the sequence of tokens, words, or sentences being generated.

•The **Constraint Relation R** is the set of rules that define the legitimate generation space: what topics are permitted, what structures are valid, what facts are confirmed, what style is appropriate.
•**Constraint Strength λ** controls how strictly these rules are enforced at each step of generation.
•A **Feedback Signal δ** is generated whenever the current output approaches or exceeds the constraint boundary, triggering real-time recalibration.

The architectural claim is: by making λ dynamic and R expandable (through the consolidation of emergent patterns), we can build a generative system that is both creative and reliably bounded.

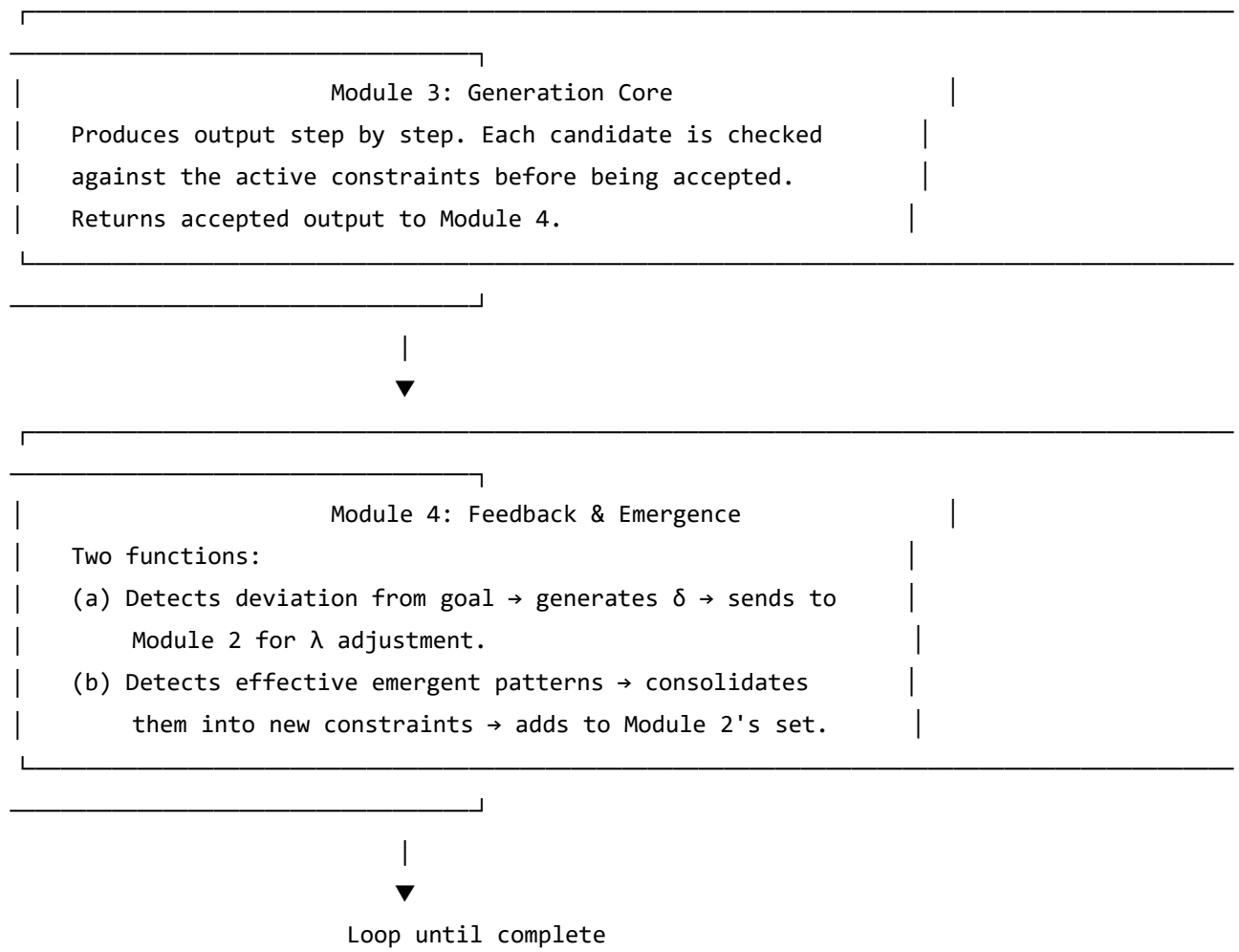
---

3. The Architecture: A Constraint-First Generation Framework

3.1 High-Level Overview

The architecture consists of four interconnected modules:

```
 text
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│                      Module 1: Goal Encoder                                                  │
│   Converts the top-level semantic goal into an initial set                                   │
│   of constraints: scope, relevance filter, style, structure                                  │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
                                          │
                                          ▼
┌──────────────────────────────────────────────────────────────────────────────────────────────┐
│                      Module 2: Constraint Engine                                             │
│   Maintains the active constraint set R(t) and the current                                   │
│   Constraint Strength λ(t). Dynamically adjusts λ based on                                   │
│   feedback signals from Module 4.                                                            │
└──────────────────────────────────────────────────────────────────────────────────────────────┘
                                          │
                                          ▼
```



3.2 Module Details

Module 1: Goal Encoder

The goal encoder takes a high-level task description and produces an initial constraint set. This is not "prompt engineering"—it is a structured transformation:

```
text
```

```
Goal Encoder:
  Input: Task description (e.g., "Write a 300-word abstract summarizing paper X")
  Process: Map to structured constraint fields
  Output:
    - Scope: [minimum length, maximum length], topic domain, required sections
    - Relevance filter: what types of information are on-topic vs. off-topic
    - Style: formal/informal, technical/popular, tone
    - Structural template: expected organization (if any)
    - Initial λ: a default Constraint Strength (e.g., 0.7)
```

Module 2: Constraint Engine

The constraint engine is the dynamic regulatory core of the system. It maintains the active constraint set R(t) and the current constraint strength λ(t), and adjusts λ in real time based on feedback signals from Module 4:

•When δ(t) = 0 or near zero, λ remains low or decays, allowing the generator to fully explore the legitimate space.
•When δ(t) > 0, λ increases for constraints relevant to the detected drift, pulling the generation back into bounds.

The adjustment of λ follows a control law (e.g., proportional-integral control), the specific form of which is a subject of empirical optimization rather than a priori prescription. The key point is: λ is **responsive**, not fixed; constraints intensify when needed and relax when safe.

Module 3: Generation Core

The generation core is the token/segment generator. Unlike standard autoregressive models, it operates under continuous constraint supervision:

```
text

Generation Step (at time t):
    1. Given current partial output S(t-1), generate k candidate continuations
       (k = 1 in greedy decoding; k > 1 for beam search with constraint scoring).
    2. For each candidate c_j:
       a. Check against all active constraints in Module 2.
       b. Compute a constraint satisfaction score:
          SCORE(c_j) = Σ w_i · r_i(c_j)
          where r_i returns 1 (valid) or 0 (invalid) for hard constraints,
          and a continuous score for soft constraints.
    3. Select the candidate with the highest combined score:
       SELECT = argmax( P(c_j | S(t-1)) + γ · SCORE(c_j) )
       (P is the base model's token probability; γ controls the constraint influence)
    4. Output the selected candidate and proceed.
```

Note: This is different from standard constrained decoding (which only filters during generation). Here, constraints also influence token probabilities at each step, not just at the final output.

Module 4: Feedback & Emergence

This is the most novel module. It performs two functions:

4a. Deviation Detection (Feedback Signal δ)

At each generation step, the system computes a semantic distance between the current partial output and the top-level goal:

$$\delta(t) = d(\text{Embed}(S(t)), \text{Embed}(\text{Goal})) - \text{threshold}$$

where:

•d is a distance metric (e.g., cosine distance in embedding space)
•threshold is a hyperparameter (the "boundary" of acceptable deviation)
•δ > 0 means the output is drifting beyond the acceptable range

When δ > 0:

1.Send δ to Module 2 (Constraint Engine).
2.λ is increased for constraints relevant to the detected drift.
3.Optionally: regenerate the last N tokens with higher λ.

4b. Emergence Detection & Consolidation

This mechanism monitors the generated output for recurring, effective patterns that were not pre-specified:

```
text

Emergence Detection:
    • Maintain a buffer of the last M tokens/segments.
    • Extract frequent patterns (n-grams, syntactic structures, argument patterns).
    • For each pattern p, compute:
       frequency(p) = count(p in buffer) / M
       effectiveness(p) = success_score(p)  (e.g., human feedback or coherence metric)
    • If frequency(p) > f_threshold AND effectiveness(p) > e_threshold:
        This pattern is considered "emerged."

Consolidation:
    • Convert the pattern into a constraint function r_new.
    • r_new checks whether future outputs conform to this pattern.
    • Add r_new to R_set with an initial λ (e.g., 0.5).
    • The system now has a new rule: "prefer this effective pattern."
```

This is the key mechanism for recursive improvement: the system learns from its own successful output and turns it into a structural constraint for future generation.

3.3 The Recursive Cycle

The complete generation process is a recursive cycle:

```
 text

START
  ↓
[1] Encode goal → initial constraints (R_initial, λ_initial)
  ↓
[2] Generate candidate token(s) under R(t), λ(t)
  ↓
[3] Check δ (deviation)
  ├── If δ > 0: increase λ → go back to [2] (or regenerate)
  └── If δ ≈ 0: continue
  ↓
[4] Check for emergent patterns
  ├── If effective pattern detected: consolidate into new constraint → add to R(t+1)
  └── If none: continue with existing R(t)
  ↓
[5] Output token(s); update S(t)
  ↓
[6] If not complete: go to [2]
  ↓
END
```

3.4 On Creativity and Constraint: A Structural Clarification

A recurring concern in constraint-based generation is the alleged trade-off between constraints and creativity: the intuition that more constraints necessarily imply less creative output. This intuition is plausible under a specific, narrow definition of creativity—namely, creativity as **unbounded novelty**, where the value of generation is equated with its distance from the familiar and its resistance to any predetermined structure.

We reject this definition as both conceptually confused and practically inadequate for engineered AI systems.

3.4.1 Creativity Is Always Domain-Bounded

In every meaningful human practice, creativity operates within constraints. A sonnet must obey meter and rhyme; a scientific paper must adhere to evidentiary standards and citation conventions; a legal argument must conform to precedent and statutory interpretation; a chess move must respect the rules of the game. In none of these cases do the constraints "reduce creativity"—they constitute the very conditions under which creativity is recognizable as such. A sequence of random words is not a poem; a collection of unsupported assertions is not a scientific argument. Constraints do not oppose creativity; they **enable** it by providing the structure within which creative acts acquire meaning, evaluation, and purpose.

Generative AI is no exception. When we ask a model to summarize a paper, we do not want "creative" fabrications; we want accurate, concise, well-structured summaries. When we ask for a safe response, we do not want "creative" boundary violations; we want outputs that remain within safety specifications. The value of a generated output is not a function of its novelty alone, but of its novelty **within the relevant constraints**. The constraints define the task. To generate outside them is not to be more creative—it is to fail at the task.

3.4.2 Constraint Strength Is Dynamic, Not Static

Even if one accepts that constraints define the task, a separate concern remains: does a high constraint strength λ reduce the expressive diversity of the generator? This concern conflates two very different control regimes.

In a **static** constraint regime, the model is permanently subjected to a fixed set of restrictions that apply uniformly across all generation steps. Under such a regime, diversity may indeed be suppressed—much like reducing the temperature parameter in standard sampling uniformly lowers the entropy of all token choices.

Our framework, however, implements a **dynamic** constraint regime. The constraint strength $\lambda(t)$ is adjusted in real time based on the feedback signal $\delta(t)$:

•When $\delta(t) \approx 0$ (the output is securely within the legitimate space), $\lambda(t)$ decreases or remains low, allowing the model's underlying probability distribution to express its full generative diversity.

•When $\delta(t) > 0$ (the output approaches or crosses the boundary), $\lambda(t)$ increases for the relevant constraints, actively pulling the generation back into bounds.

This is the functional equivalent of a **guardrail** on a mountain road: it does not prevent the vehicle from navigating curves, changing lanes, or adjusting speed; it only prevents the vehicle from leaving the road entirely. The driver retains full control within the safe zone. Similarly, the generator retains—and in fact exercises—its creative capacity across the entire legitimate space. Constraint enforcement is localized and responsive, not global and suffocating.

3.4.3 The Emergence–Locking Cycle Expands Competence, Not Restrictions

The third objection concerns the consolidation mechanism: does locking emergent patterns into new constraints progressively shrink the generation space over time, leading to a rigid, repetitive system?

This objection, again, rests on a misinterpretation. When an emergent pattern is consolidated into a constraint, the system does not **prohibit** all other patterns. Rather, it **privileges** a pattern that has proven effective for the current task. The constraint function r_new checks for conformity to this successful pattern, but it does not actively suppress alternatives; it simply adds a positive score contribution for future outputs that resemble it.

Moreover, as noted in Section 3.4.2, the enforcement weight of any given constraint is modulated by $\lambda(t)$, which decays when the output remains stable. A consolidated constraint with initial $\lambda = 0.5$ may, over the course of a generation, see its effective enforcement rise or fall depending on the output's behavior. If the generator later discovers an even more effective pattern, that new pattern will be consolidated as well—and the system will possess a richer, not poorer, repertoire of successful strategies.

This is precisely how human expertise develops. A novice chess player explores random moves; a master has locked thousands of patterns into intuition. Those locked patterns do not reduce the master's creativity—they free cognitive resources for higher-order strategic thinking. The same logic applies here: locking is the mechanism by which a system accumulates intelligence, not the mechanism by which it accumulates sclerosis.

3.4.4 Summary: A Framework for Bounded Creativity

To summarize the position of this framework on creativity:

1. **We do not value unbounded creativity.** In engineered AI systems, the ability to produce hallucinated falsehoods, incoherent arguments, or unsafe content is not a feature—it is a failure mode to be structurally eliminated.
2. **We value task-relevant creativity.** The legitimate space defined by the top-level constraint set is precisely the region within which meaningful creative variation can occur. The architecture actively preserves the generator's diversity within this space.
3. **We implement creativity-preservation dynamically.** Constraint strength λ responds to the output's actual behavior in real time, tightening only when necessary and relaxing when safe, ensuring that constraint enforcement does not uniformly suppress expression.
4. **We treat the emergence–locking cycle as competence accumulation.** Locking effective patterns into new constraints builds the system's structured knowledge over time, expanding its capacity for successful bounded creativity rather than shrinking its expressive range.

---

4. Comparison with Existing Approaches

Approach	Constraint Timing	Constraint Type	Emergence Handling	Interpretability
Standard LLM	None (post-hoc filtering)	Implicit (statistical)	None	Low
RLHF	Post-training	Soft (reward model)	None	Low
Constitutional AI	During fine-tuning	Rule-based, soft	None	Medium
Constrained Decoding	During generation	Hard filters	None	Medium
Prompt Engineering	Pre-generation	Soft (instruction)	None	Low
This Framework	**Before + During generation**	**Hybrid (hard + dynamic soft)**	**Consolidated into new constraints**	**High**

Key differentiators of this framework:

1. **Constraints are applied before generation begins** (the goal encoder defines the legitimate space a priori), not just during or after.

2.**Constraint strength is dynamic** (λ adjusts in real time based on feedback signals), not static.
3.**Emergence is captured and consolidated** (effective new patterns become constraints), not ignored or treated as unpredictable.
4.**Generation is traceable** (the constraint history explains why each part of the output was generated), not a black box.

A subtle but crucial distinction deserves explicit attention. In standard LLM decoding, "creativity" is operationalized as **high-entropy sampling**—the model is more likely to choose low-probability tokens, which increases novelty but also increases the risk of hallucination and incoherence. Within our framework, creativity is operationalized as **high-entropy sampling within a constrained space**. The constraint set defines the boundaries of the task; within those boundaries, the model retains full access to its generative diversity. The difference is not the absence of temperature-like exploration, but the presence of a **boundary-enforcing control loop** that prevents exploration from escaping the task-relevant region. This is the difference between a writer working within a genre and a writer producing random strings—both are "creative" in a literal sense, but only the former is useful.

---

5. Implementation Considerations

5.1 Constraint Representation

Constraints can be represented in multiple forms, depending on the generation level:

Level	Constraint Type	Implementation
Token-level	Lexical/grammatical	Vocabulary restrictions, grammar rules, token-probability masking
Sentence-level	Semantic/relevance	Embedding similarity threshold, topic classification
Structural-level	Organizational	Template matching, section-ordering rules
Goal-level	Topical/factual	Knowledge base verification, fact-checking modules

5.2 The Base Model

This framework is **model-agnostic**. It can be implemented on top of any autoregressive language model (Llama, GPT, Gemini, etc.) by:

1.Intercepting the token generation step.
2.Injecting constraint scoring and selection logic.
3.Adding the feedback and emergence modules as external control loops.

The base model's next-token probabilities are **augmented**, not replaced, by constraint scores.

5.3 Emergence Detection in Practice

A practical implementation of emergence detection can use:

•**Statistical n-gram analysis:** Identify frequent sequences.
•**Entropy reduction:** Monitor if a pattern reduces the entropy of subsequent generations (i.e., it "stabilizes" the output).
•**External evaluation:** Use a separate model to score coherence, factuality, or task-specific metrics; if a pattern is consistently associated with high scores, consolidate it.

5.4 Computational Overhead

Compared to standard generation, this framework introduces additional computational costs:

•**Constraint checking:** O(m) per token where m = number of active constraints
•**Emergence detection:** O(M·K) per step where M = buffer size, K = pattern extraction complexity
•**Feedback computation:** O(embedding dimension) per step

These are significant but not prohibitive. A typical system with 10-30 constraints and a 100-token buffer should be feasible on modern hardware. Optimizations include:

•Caching constraint evaluation results
•Using sparse pattern extraction
•Running emergence detection periodically (every N steps), not every step

---

6. Potential Use Cases

6.1 Hallucination Reduction

Problem: LLMs frequently generate false facts that are fluent but incorrect.

Application: The top-level constraint defines "output must be grounded in the provided knowledge base." Constraint checking includes a fact-verification module. δ is triggered when the model generates content that diverges from the knowledge source. λ is increased for factual constraints. Emergent patterns that correlate with high factuality are consolidated as new constraints.

6.2 Long-Form Logical Consistency

Problem: Long generations often contradict themselves or lose the thread.

Application: The top-level constraint defines "maintain the argument structure." A "context memory" constraint tracks key claims made earlier; if a later claim contradicts an earlier claim, δ increases and λ for the consistency constraint increases. Emergent rhetorical strategies that maintain coherence are consolidated.

6.3 Safe/Constitutional Generation

Problem: Models generate harmful or biased content.

Application: The top-level constraint defines "all output must be within safety boundaries." A set of safety classifiers serve as hard constraints. δ is triggered when the model approaches a safety boundary. This is a more dynamic and responsive version of Constitutional AI, with real-time adjustment rather than post-hoc fine-tuning.

6.4 Creative but Bounded Generation

Problem: High-temperature generation is creative but often incoherent; low-temperature generation is coherent but boring.

Application: The top-level constraint defines "creative output within a coherent theme." Creativity is encouraged by starting with moderate λ and allowing it to decrease when δ is near zero. When λ is low, new patterns emerge; when they are effective, they are consolidated—so the system gets "smarter" and more coherent over time without losing creativity.

---

7. Limitations and Open Questions

1. **Constraint definition remains manual.** The goal encoder currently requires a human to specify what counts as "within scope" or "factually correct." Automated constraint extraction from task descriptions is an open research problem.
2. **Emergence detection is heuristic.** Our proposed detection method (frequency + effectiveness) is plausible but untested. The shape of the consolidation function—when exactly a pattern becomes a constraint—is a design choice that requires empirical validation.
3. **Constraint conflicts.** When multiple constraints conflict (e.g., "be creative" vs. "be factual"), the arbitration mechanism (weighted combination vs. priority) needs to be defined per use case. There is no general solution yet.
4. **Computational scalability.** For very long generations or very large constraint sets, the overhead may become significant. Optimizations and approximate constraint checking will likely be needed.
5. **Empirical validation.** This architecture is currently a theoretical proposal. We have not yet implemented it or tested it on real systems. Empirical work is the necessary next step.
6. **Distinguishing strategic deviation from genuine drift.** The semantic distance metric d(Embed(S), Embed(Goal)) used to generate the feedback signal δ does not currently distinguish between outputs that deviate from the goal as a strategy (e.g., temporarily exploring a counter-argument in order to refute it) and outputs that genuinely drift away from the task. This distinction requires more sophisticated intent modeling beyond current embedding-based similarity measures, and remains an open challenge for future work.

---

8. Conclusion

This white paper has presented a **constraint-first architecture for generative AI**. It is grounded in the "1=N" hierarchical framework but does not require familiarity with it to be understood.

The core claims are:

1. **The current generation paradigm is structurally flawed.** It applies constraints after generation, which is too late to prevent errors. Constraints should be applied before and throughout generation.
2. **A dynamic, closed-loop control system—with Constraint Strength λ and Feedback Signal δ—can maintain generation within a legitimate space while allowing creativity within it.**
3. **Effective emergent patterns should be consolidated into new constraints.** This creates a recursive system that improves its own constraints over time.
4. **The architecture is implementable on top of existing LLMs** without requiring retraining from scratch.
5. **The framework offers a unified approach** to hallucination reduction, consistency maintenance, safety enforcement, and controlled creativity.

This is not a fully solved system—it is a blueprint. It identifies the structure of the problem and proposes a general solution. The engineering details, empirical validation, and refinements are the next phase of work.

We invite collaboration and encourage empirical testing.

Acknowledgments

The author acknowledges the use of large language models (LLMs) as assistive tools in the preparation of this manuscript. Specifically, LLMs were employed for language polishing, grammar correction, and formatting assistance. All conceptual foundations, theoretical arguments, architectural design, and formal definitions presented in this paper are the sole intellectual contributions of the author. The core thesis—that generation should proceed within a priori constraints rather than being filtered post-hoc, and that creativity must be understood as bounded by task-defining constraints—originates from and remains the

author's own theoretical position. The author takes full responsibility for the accuracy, validity, and intellectual integrity of all claims made herein.

References

[1] Gao, Z. (2025). The 1=N Principle: A Hierarchical Ontology of Functional Locking. Zenodo. https://doi.org/10.5281/zenodo.18420680

[2] Gao, Z. (2026a). The Possible vs. The Impossible: Two Directions of Observing System Existence [Preprint].

[3] Gao, Z. (2026b). How Function Gets Locked: A Case Study of a Bottle and a Hand [Preprint].

[4] Gao, Z. (2026c). Scope-Precision and Point-Precision: A Justification of Holistic Knowledge [Preprint].

[5] Gao, Z. (2026d). Vertical Coupling: The Unit of Analysis for Boundary Logic [Preprint].

[6] Gao, Z. (2026e). Constraint Strength and Feedback Signals: A Dynamical Tool for Boundary Logic [Preprint].

[7] Anderson, P. W. (1972). More is Different. Science, 177(4047), 393-396.

[8] von Bertalanffy, L. (1968). General System Theory: Foundations, Development, Applications. George Braziller.

[9] Wiener, N. (1948). Cybernetics: Or Control and Communication in the Animal and the Machine. MIT Press.

[10] Simon, H. A. (1962). The Architecture of Complexity. Proceedings of the American Philosophical Society, 106(6), 467-482.

[11] Rosen, R. (1991). Life Itself: A Comprehensive Inquiry into the Nature, Origin, and Fabrication of Life. Columbia University Press.

Appendix: Glossary of Key Terms

Term	Definition
Constraint	A top-down logical boundary that delimits the legitimate range of system behavior
Constraint Strength (λ)	The degree to which a constraint is enforced, ranging from 0 (none) to 1 (fully enforced)
Feedback Signal (δ)	A signal generated when generation deviates from the top-level goal; intensity proportional to deviation
Emergence	New structures or patterns arising from lower-level interactions
Locking	The process of solidifying an emergent pattern into a constraint
Goal Encoder	Module that converts a task description into an initial constraint set
Constraint Engine	Module that maintains and dynamically adjusts active constraints
Generation Core	The token/segment generator that operates under constraint supervision
Emergence Detector	Module that identifies effective recurring patterns and consolidates them into constraints
Constraint History	A trace of all constraints active during generation, enabling interpretability