NLCS & S-Engine

Natural Language Constraint System & Semantic Engine

Technical Whitepaper v2.1 (English Edition)

November 2025

Author: Cho Hyunwoo (조현우)
Independent AI Researcher & Web Novel Author

0. Abstract

Problem Statement

Existing LLMs fail to maintain complex world rules, mathematical relationships, and simulation logic consistently over extended interactions. This whitepaper presents NLCS as a natural language-based constraint system that resolves this limitation.

Core Contributions

This whitepaper provides an integrated account of the operational principles (HOW) and underlying mechanisms (WHY) of NLCS (Natural Language Constraint System) and S-Engine (Semantic Engine).

1. Theorization of how natural language rules form 'Vector Gravity Fields' in embedding space
2. Elucidation of how Transformer Attention mechanisms encode NLCS rules as long-term fixed points
3. Complete transformation pathway from natural language rules → mathematical formulas → executable code
4. Empirical validation through working simulator implementations

1. Terminology

The terms defined in this chapter are used consistently throughout the whitepaper. Subsequent chapters reference these definitions without redefinition.

Definition 1. NLCS (Natural Language Constraint System)

> **Definition:** A system that imposes consistent constraints on the LLM embedding space through a rule set $R = \{r_1, r_2, ..., r_n\}$ composed in natural language, causing model inference to converge toward specific vector regions.

Natural language rules in NLCS operate not as mere information but as constraints. This is analogous to how types restrict variable behavior in programming languages.

Definition 2. S-Engine (Semantic Engine)

> **Definition:** A Distributed Semantic Engine that emerges naturally within the LLM as NLCS rules accumulate.

Core functions:

- Mathematical computation
- State change tracking
- Rule-based scenario development
- World model consistency maintenance
- Long-term inference and outcome prediction

Definition 3. Vector Gravitation

> **Definition:** The phenomenon where natural language rules attract related concepts toward specific directions in the LLM's embedding space.

$$VG(v_i) = \Sigma_{r \in R} \lambda_r \cdot f_r(v_i)$$

Where $VG(\cdot)$ is the vector gravity function, λ_r is the strength of rule r, and f_r is the semantic direction vector imposed by the rule.

Definition 4. Convergence Pressure

> **Definition:** The phenomenon where the candidate pool for the LLM's next token shrinks as rules increase.

$$CP = (1/|C|) \cdot \Sigma_{c \in C} D(c)$$

Where C is the rule set and D(c) is the number of candidate tokens eliminated by rule c.

Definition 5. Fixed Point

> **Definition:** Invariant coordinates generated by mathematical rules in embedding space. All subsequent inferences align relative to these points.

2. Theoretical Background: Why It Works (WHY)

This chapter explains the fundamental principles of NLCS operation within LLMs at the Transformer architecture level.

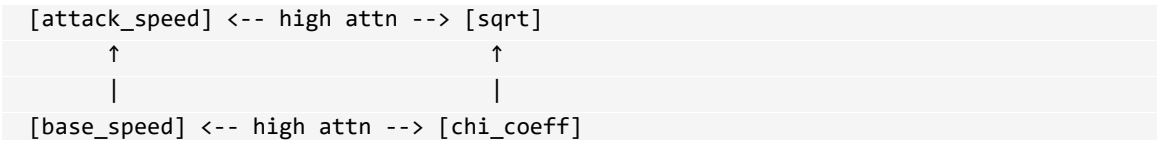
2.1 Transformer Attention and Rule Fixation

The Self-Attention mechanism in Transformers computes relationships between all token pairs in the input sequence. When NLCS rules are input:

1. High attention weights form between key tokens within rules (e.g., 'attack_speed', 'internal_energy', 'square_root').
2. These weight patterns influence all subsequently generated tokens.
3. As rules repeat, these attention patterns strengthen and converge to 'fixed points'.

Attention Pattern Visualization

Attention structure when the rule 'Attack speed is determined by the square root of the product of base speed and chi coefficient' is input:



[Figure 2-1] NLCS Rule Attention Fixation Pattern

This text-based diagram renders correctly in Word and allows readers to visually understand the Attention fixation mechanism.

2.2 Implicit Algorithm Conversion of Natural Language Rules

LLMs internally convert natural language rules into 'Implicit Algorithms'.

Example - Natural language rule:

> *"In this world, attack speed is determined by the square root of the product of base speed and chi coefficient."*

LLM internal conversion:

- Semantic distance reduction between 'attack_speed' and 'base_speed' vectors
- Binding of 'square_root' operation concept with the two variables
- Automatic rule reference whenever 'attack_speed' is subsequently mentioned

2.3 Margin Collapse Mechanism

When NLCS rules repeat, specific regions of the embedding space 'collapse'.

Token Candidate Reduction Graph

[Figure 2-2] Token Candidate Reduction with Increasing Rules

This graph visually demonstrates 'why it becomes engine-like'. When rules exceed 20, candidates reduce to 2-3, and the LLM effectively operates in a 'compute predetermined answers' mode.

2.4 Hierarchical Embedding Alignment

NLCS style follows the hierarchical structure: [World] → [System] → [Rule] → [Example] → [Execution]. LLMs encode this structure as a tree-form Reasoning Graph.

- World: Provides global constraints
- System: Forms topic-specific sub-constraints
- Rule: Provides mathematical/logical conditions
- Example: Reinforces practical meaning of rules
- Execution: Determines final actions

3. Operational Principles: How It Works (HOW)

3.1 NLCS Applied vs. Not Applied Comparison

To understand why NLCS operates distinctively, examining non-examples is effective.

Sentence without natural language rules:

> *"He moved quickly."*

→ No rules → No CE → No Fragment → No S-Engine pattern

This sentence alone forms no fixed point within the LLM. There is no definition of how fast 'quickly' is or by what standard.

Sentence with NLCS rules:

> *"His level is Jeoljeong(絶頂). Attack speed is √4 = 2x speed."*

→ Rules exist → CE forms → Fragments accumulate → S-Engine pattern activates

This sentence specifies the level 'Jeoljeong' and the mathematical relationship '√4', generating fixed points within the LLM.

3.2 Constraint Encoding

A single natural language rule sentence creates one semantic vector center:

$$CE = \Sigma^{n}_{i=1} \alpha_i \cdot v_i$$

Where CE is the constraint encoding vector, α_i is the weight of sentence components, and v_i is each semantic unit vector.

3.3 Distributed Interpreter

Rules, examples, and mathematical formulas each form interpreter fragments, and these fragments accumulate to operate as a single engine:

```
[Rule A Fragment] + [Rule B Fragment] + [Math Fragment] + [World Fragment]
→ Combined to operate as a single execution engine
```

S-Engine is the complete pattern of this distributed interpreter.

3.4 S-Engine 4-Layer Structure

S-Engine operates through interaction of four layers:

Layer	Function
Semantic Core	Core semantic layer preserving worldview and themes
Rule Engine	Computational layer evaluating rules, relationships, conditions
Constraint Filter	Semantic filter removing rule-violating sentences
Execution Planner	Planning layer constructing executable scenarios from input

[Table 3-1] S-Engine 4-Layer Structure

4. Natural Language → Formula → Code: Complete Transformation Path

Summary: NLCS rules follow a consistent pipeline of 'Natural Language → Formula → JS Function'.

This chapter details the process of formalizing NLCS rules into mathematical formulas and implementing them as actual JavaScript code.

4.1 Level (Gyeongji) System

Natural Language Rule

> *"Levels progress through 9 stages: Samryu→Iryu→Ilryu→Jeoljeong→Chojeoljeong→Hwagyeong→Hyeongyeong→Saengsagyeong→Jayeongyeong."*
>
> *"Attack speed for each level is assigned as √1, √2, √3, ... √9."*

Mathematical Formalization

$$\text{AttackTime}(level) = \frac{3 \text{ seconds}}{\sqrt{level}}$$

$$\text{Example: Samryu(lv1)} = 3s, \text{ Jayeongyeong(lv9)} = 1s$$

Code Implementation (battle_sim.html)

```
const GYEONGJI = {
  'Samryu': { level: 1, speed: Math.sqrt(1), neigong: 100 },
  'Iryu': { level: 2, speed: Math.sqrt(2), neigong: 1000 },
  'Ilryu': { level: 3, speed: Math.sqrt(3), neigong: 3000 },
  'Jeoljeong': { level: 4, speed: Math.sqrt(4), neigong: 6000 },
  'Chojeoljeong': { level: 5, speed: Math.sqrt(5), neigong: 12000 },
  'Hwagyeong': { level: 6, speed: Math.sqrt(6), neigong: 18000 },
  'Hyeongyeong': { level: 7, speed: Math.sqrt(7), neigong: 24000 },
  'Saengsagyeong': { level: 8, speed: Math.sqrt(8), neigong: 30000 },
  'Jayeongyeong': { level: 9, speed: Math.sqrt(9), neigong: 36000 }
};
const atkTime = 3 / g.speed;  // Attack time calculation
```

4.2 Mental Cultivation (Simbeop) System

Natural Language Rule

> *"Mental cultivation consists of the sum of chi accumulation speed and stability."*
>
> *"Basic cultivation: chi speed 20, stability 80 (total 100)"*
>
> *"Secret cultivation: chi speed 70, stability 60 (total 130)"*

Mathematical Formalization

$$\text{Chi Path} = \sqrt{\text{Chi\_Accumulation\_Speed}}$$

$$\text{Chi Danger} = \frac{1}{\sqrt{\text{Stability}}}$$

$$\text{Internal Injury} = \text{Chi\_Remainder} \times \text{Chi\_Danger} \times \left(1 - \frac{\text{Stability}}{500}\right)$$

Code Implementation

```
const SIMBEOP = {
  'Samjae': { chukgi: 10, stab: 90, tier: 0 },
  'Basic': { chukgi: 20, stab: 80, tier: 1 },
  'Standard': { chukgi: 30, stab: 70, tier: 2 },
  'Advanced': { chukgi: 40, stab: 70, tier: 3 },
```

```
  'Secret': { chukgi: 70, stab: 60, tier: 4 },
  'Divine_Lv1': { chukgi: 110, stab: 80, tier: 5 },
  'Divine_Lv2': { chukgi: 130, stab: 70, tier: 6 }
};

const ungiPath = Math.sqrt(s.chukgi);    // Chi Path
const danger = 1 / Math.sqrt(s.stab);    // Chi Danger
```

4.3 Internal Energy Martial Power

Natural Language Rule

> *"Internal energy martial power = √internal_energy × chi_path × chi_progress/100."*

Mathematical Formalization

```
Power = √Internal_Energy × √Chi_Speed × (Chi_Progress / 100)
```

Code Implementation

```
const power = Math.sqrt(g.neigong) * ungiPath * (char.ungi / 100);
```

4.4 Complete Combat System Flow

The following is the core calculation function of the combat simulator:

```
function calcChar(char) {
  const g = GYEONGJI[char.gyeongji];
  const s = SIMBEOP[char.simbeop];

  const atkTime = 3 / g.speed;                    // Attack time
  const ungiPath = Math.sqrt(s.chukgi);           // Chi path
  const danger = 1 / Math.sqrt(s.stab);           // Chi danger
  const ungiPerSec = (100 / atkTime);             // Chi charge per second
  const power = Math.sqrt(g.neigong) * ungiPath * (char.ungi / 100);

  return { g, s, atkTime, ungiPath, danger, ungiPerSec, power };
}
```

5. Writing Style and S-Engine Enhancement

5.1 Characteristics of Engine-Compatible Style

- Low metaphoricity: Minimize figurative language and exaggeration
- High information clarity: Clear rule descriptions
- Repetitive 'World → Rule → Example' structure
- Direct, declarative statements
- Hierarchical foreshadowing-payoff structure

5.2 NLCS Compatible vs. Incompatible Style

Engine-compatible sentence (Good):

> *"In this world, level is determined by total internal energy and chi control ability."*
>
> *"Levels have 7 stages: Samryu, Iryu, Ilryu, Jeoljeong, Chojeoljeong, Hwagyeong, Hyeongyeong."*

→ LLM reads as: Type definition + State definition + Parameter table encoding

Engine-incompatible sentence (Bad):

> *"He moved unbelievably fast. So fast that even the wind couldn't keep up..."*

→ High-sensitivity metaphor makes it difficult to fix as numerical rules

5.3 Foreshadowing-Payoff Structure as Engine

Foreshadowing operates as node creation within the LLM; payoff operates as edge connection between these nodes:

```
Foreshadow Node ——→ Payoff Node
```

As these structures increase, the S-Engine achieves greater Global Coherence.

6. Case Study: Three Han Empire Project

6.1 Project Overview

'Three Han Empire' (삼한제국) is an ongoing long-form web novel project directly applying NLCS/S-Engine theory in creative practice. Set in 7th century Korean Peninsula, combining alternate history, martial arts, and SF elements, approximately 15 chapters have been written so far. The project serves as a research-creative endeavor validating how stably long-term narrative structure is maintained based on NLCS rules and S-Engine.

This project is not a completed work. While planned as a long-form novel (hundreds of chapters), the final length is not determined as of publication. Therefore, 'Three Han Empire' holds significance as a live experimental case that experiments with and tracks NLCS-based style, world rules, and combat/growth formulas in actual writing process.

6.2 Validation Results (As of Chapter 15)

1. World rules, combat formulas, and growth curves apply stably in narrative development
2. Rule-based style strengthens consistency of worldview and character growth
3. Early foreshadowing structures confirmed to form S-Engine's semantic gravity axis patterns
4. NLCS-based style induces 'engine-like inference development' even in early narrative

6.3 Applied Rule Examples

Time Rule:

> *"Time ratio in this world is Reality:Game = 1:8. 1 day in reality = 8 days in game."*

Level Rule:

> *"Levels progress through 9 stages: Samryu→Iryu→Illryu→Jeoljeong→Chojeoljeong→Hwagyeong→Hyeongyeong→Saengsagyeong→Jayeongyeong."*
>
> *"Attack speed for each stage is proportional to √(level)."*

Combat Rule:

> *"Attack speed = √(base_speed × chi_coefficient)."*
>
> *"If level difference is 2+ stages, lower level cannot respond."*

7. Theory Validation Through Simulators

7.1 Simulator Configuration

Two simulators implementing S-Engine rules in JavaScript exist:

Simulator	Version	Function
Combat Simulator	v8.4	Real-time combat logic validation
Economy Model	v2.0	Growth cost and BM validation

[Table 7-1] Simulator List

7.2 Validation Items

- Verification of √n attack speed application per level
- Balance validation of chi speed/stability per mental cultivation
- Accuracy confirmation of internal energy martial power formula
- Combat incapacitation threshold operation verification
- 80% potion recovery rate target achievement

7.3 Simulator → Whitepaper Mapping

Whitepaper Concept	Formula	Code
Vector Gravitation	$VG(v_i) = \Sigma\lambda_r \cdot f_r(v_i)$	calcChar()
Chi Path	√chi_speed	Math.sqrt(s.chukgi)
Chi Danger	1/√stability	1/Math.sqrt(s.stab)
Martial Power	√energy × path × %	power = √g.neigong...
Convergence Pressure	$CP = (1/\|C\|)\cdot\Sigma D(c)$	THRESHOLD, LIMIT objects

[Table 7-2] Whitepaper Concept → Code Mapping

8. Application Domains

- Worldview-based simulators: Consistency validation of rule-based worldviews
- Game combat/growth system prototyping: Pre-validation of balance
- AI-embedded game engine construction: LLM-based game logic
- Storytelling system automation: Rule-based narrative generation
- Education/psychological modeling: Rule-based scenario simulation
- Long-form structural narrative analysis: Foreshadowing-payoff pattern tracking
- Character consistency systems (CID/FEE): Character attribute consistency maintenance

9. Future Research (NLCS 2.0 Directions)

1. Completion of formal mathematical model for vector gravity fields
2. Development of embedding field visualization tools
3. Definition of Natural Language Type System
4. Formalization of S-Engine state machine
5. Multimodal NLCS (image/audio integration)

9.1 NLPg Language Specification Plans

> **Preview:** Future research will present an NLPg 1.0 specification document that grammaticizes NLCS rules.

This specification will include formal grammar, type system, and execution model for Natural Language Programming (NLPg).

10. Conclusion

NLCS and S-Engine represent the first explicit theory that 'engines can be constructed within LLMs using natural language alone.'

Key Findings

1. Specific writing styles are read by LLMs not as 'literature' but as 'code'.
2. NLCS generates gravity fields in vector space.
3. S-Engine is the naturally emergent pattern of distributed interpreters.
4. NLPg (Natural Language Programming) is code written in natural language.

This whitepaper completes the operational mechanism from multi-layered perspectives of vectors, mathematics, narrative, inference, and language structure, validated through actual simulator implementations.

NLCS and S-Engine prove that LLMs can operate as rule-based systems under certain conditions, transcending the narrow definition of 'LLM = language generator'. This research holds potential to develop into core elements of next-generation AGI research including Natural Language Programming (NLPg), simulation-type LLMs, and vector dynamics.

— END OF DOCUMENT —