Abstract

With the emergence of GPT-based models, humanity has entered the era of semantic intelligence—a paradigm shift in computer science from fixed algorithms to adaptive and generative ones.

As large language models (LLMs) rapidly expand their capabilities in generation and reasoning, their influence on knowledge-intensive industries continues to grow.

However, the black-box nature and lack of traceability of LLMs limit their full potential in fields that require interpretability, verifiability, and accountability.

To address these challenges, many AI organizations have established explainability divisions to make LLM reasoning more transparent and evidence-based.

Meanwhile, the phenomenon of hallucination continues to create uncertainty for both users and developers.

In response, we propose a new universal language for structured knowledge representation—Semantic Tension Language (STL)—designed to bridge the gap between natural language flexibility and formal logical precision.

STL provides a human-readable yet machine-executable syntax for representing knowledge with explicit semantic relations, contextual attributes, and reasoning transparency.

1 – Introduction

Since ancient times, humanity has used various methods to encode and transmit information—from beacon fires on city walls to the rhythmic signals of war drums [1] [2]. These primitive systems represent the earliest attempts to map external reality into symbolic form. In modern times, natural language serves as humanity's primary medium for constructing an internal world model—a symbolic mirror of perception and cognition.

Formal logical languages emerged as a more rigorous evolution of natural language, built upon propositions, inference, and truth values [3] [4]. They provide the theoretical foundation for mathematics, computer science, and analytic philosophy [5] [6]. Yet, with the rapid rise of large language models (LLMs), a new paradox has surfaced. The very strength of natural language—its contextual flexibility and probabilistic semantics—introduces uncertainty, ambiguity, and non-determinism into machine reasoning. Although formal logic offers precision and verifiability, it struggles to represent the

complex, multi-causal, and asymmetric relationships that characterize real-world phenomena. [7]

Therefore, natural and formal languages reveal a structural duality: one excels in semantic richness, the other in computational precision. Bridging this gap is essential for the next generation of AI systems that demand both interpretability and expressive power. [8]

In this work, we introduce the Semantic Tension Language (STL) — a universal structural language that encodes not the external world itself, but the structure of knowledge. STL transforms naturally expressed knowledge into a computable, traceable, and context-aware representation, eliminating semantic redundancy and enabling more interpretable reasoning for LLM-based systems.

2 – Related Work

2.1 Evolution of Semantic Representation Languages

The study of how knowledge can be expressed in structured form has evolved through several stages.

Early systems such as frame semantics and semantic networks sought to capture human concepts and their relationships in symbolic graphs. [9]

Later, projects such as FrameNet, PropBank, and Abstract Meaning Representation (AMR) provided more formalized ways to represent meaning at sentence and discourse levels. [10] [11] [12]

These representations encode events, agents, and roles but remain limited in expressing higher-order relations, uncertainty, or contextual modulation that are essential for complex reasoning. [13]

They primarily emphasize “what is said,” whereas emerging AI applications increasingly demand a representation of “how it is known” and “under what conditions it holds.”

While STL is not derived from existing resources such as FrameNet, PropBank, or AMR, it addresses a similar goal — to formalize the semantics of natural language — but extends the representation toward dynamic, computable knowledge structures.

2.2 Semantic Web and Knowledge Structuring Standards

The Semantic Web initiative introduced by the W3C established foundational standards—RDF, RDFS, and OWL—to formalize knowledge into machine-readable triples. [14] [15] [16]

These frameworks enabled logical inference and ontology-driven reasoning at the Web scale, laying the groundwork for linked-data ecosystems.

However, such symbolic formalisms rely heavily on predefined ontologies and binary relations.

They often struggle to model dynamic, context-dependent knowledge where meanings shift across domains or temporal frames.

Attempts to enhance expressivity—such as SHACL, PROV-O, and context-aware graph extensions—partially address these challenges but do not fully capture the fluid semantics of natural language or the probabilistic reasoning style exhibited by modern LLMs. [17] [18] [19]

2.3 Structural Divide between Symbolic and Connectionist Paradigms

A long-standing challenge in artificial intelligence lies in reconciling symbolic and connectionist approaches to knowledge representation.

Symbolic systems—rooted in logic and ontology—offer precision, traceability, and explicit reasoning, yet they struggle to handle ambiguity, contextual variation, or emergent meaning. [20] [21]

Connectionist systems, conversely, excel at adaptive pattern recognition and probabilistic generalization but lack transparent reasoning structures and verifiable semantics. [22] [23]

This divide is not merely technical but structural: symbolic systems rely on discrete formal units, while connectionist models operate on continuous distributed representations. [24]

Bridging these fundamentally different modes of representation remains one of the unresolved questions in AI theory. [8] [25]

2.4 Position of STL within Existing Paradigms

The Semantic Tension Language (STL) approaches this challenge from a different perspective.

Rather than attempting to merge symbolic and connectionist architectures at the computational level, STL proposes a linguistic and structural mediation layer that redefines how meaning itself can be represented.

Through its triadic design—Anchor, Path, and Modifier—STL formalizes both the stable skeleton of knowledge and the contextual dynamics that shape its interpretation.

This formulation offers a way to express semantic variability within a structured and traceable syntax, allowing reasoning systems to capture the interplay between logic and context without collapsing one into the other.

In this sense, STL does not claim to have solved the integration problem; instead, it provides an alternative entry point—a representational approach that could enable future systems to translate between symbolic precision and semantic fluidity.

By focusing on the tension between meanings rather than their categorical separation, STL reframes the integration challenge as a problem of representation, not architecture.

3 – Theoretical Framework of STL

This section presents the original theoretical framework of the Semantic Tension Language (STL), developed independently by the author. While the framework draws conceptual inspiration from existing representation paradigms such as RDF's triple model, all structural definitions and mechanisms introduced herein—including Anchors, Paths, and Modifiers—are original formulations designed to extend beyond the limitations of predefined ontological schemas.

3.1 Anchor System

3.1.1 Definition and Function

An Anchor is the minimal representational unit of meaning in the Semantic Tension Language (STL).

Each Anchor denotes a semantically independent node that carries cognitive reference within the knowledge structure.

Anchors may represent concepts, agents, questions, or intermediate path segments that serve as identifiable points of semantic reference.

Functionally, Anchors form both the origin and termination of all semantic paths, providing reference stability within an otherwise dynamic semantic network.

They serve as the reference points through which semantic relations are projected, interpreted, and verified.

3.1.2 Three-Level Hierarchy of Anchors

Anchors in STL can be organized into three abstraction levels according to their semantic role:

- Semantic Layer – Represents abstract concepts, logical relations, or general knowledge units; roughly corresponding to nouns and verbs in natural language.
- Entity Layer – Refers to concrete objects, persons, events, or named entities; corresponding to proper nouns or event terms.
- Structural Layer – Represents meta-linguistic or reasoning structures such as agents, cognitive roles, or logical variables.

Together, these three layers constitute the semantic space of STL, within which meaning can be expressed from abstraction to concreteness and from cognition to structure.

3.1.3 Anchor Taxonomy — Nine Core Anchor Types

Across the three layers, STL defines nine canonical categories of Anchors, forming the semantic skeleton of the representational universe.

Type	Description	Examples	Layer
Concept Anchors	Abstract ideas, theories, properties, or categories	[Freedom], [Energy], [Entropy], [Language]	Semantic
Relational Anchors	Logical or semantic relations between entities	[Cause], [Effect], [Identity], [Opposition]	Semantic
Event Anchors	Specific actions, processes, or temporal events	[War], [Conference], [Migration], [Computation]	Entity
Entity Anchors	Concrete physical or perceivable objects	[Apple], [Table], [Galaxy], [Human]	Entity
Name Anchors	Uniquely identified named entities	[Einstein], [London], [Google], [TheBible]	Entity
Agent Anchors	Active or cognitive subjects	[Self], [Researcher], [AI_System], [Society]	Structural
Question Anchors	Points of inquiry or unresolved cognitive tension	[Question], [Unknown], [Why], [Hypothesis]	Structural
Verifier Anchors	Mechanisms of evaluation, testing, or validation	[Verifier], [Test], [Criterion], [Observer]	Structural
Path Segment Anchors	Intermediate states or transitional nodes	[Process], [Transition], [Bridge], [IntermediateState]	Structural

These nine classes define the ontological backbone of STL.However, structure alone would render the system rigid and lifeless.To infuse variability, expressiveness, and

contextual nuance, STL introduces a complementary category known as Derived Anchors.

3.1.4 Derived Anchors

Definition:
Derived Anchors are secondary semantic units derived from the nine core types.

They describe multidimensional semantic features such as time, space, emotion, value, modality, and quantity.

While core Anchors form the skeleton of meaning, Derived Anchors provide its semantic texture—the dynamic qualities that enable expressiveness and contextual adaptation.

Derived Anchors may appear as independent nodes in higher-level semantic graphs or as value sources referenced within the Modifier (::mod) construct (see § 3.3).

When used as modifier parameters, they enrich meaning without disturbing syntactic purity: ::mod() never directly contains Anchors, only their attribute references.

Functions of Derived Anchors:

1. Context Control.

   Derived Anchors determine how semantic structures expand across dimensions such as time, space, emotion, and value, endowing STL with context-sensitive expressive power.

2. Attribute Projection.

   Serving as sources of semantic dimensions, Derived Anchors can project their properties onto Core Anchors or Path Expressions as ::mod() parameters, enabling the construction of multi-dimensional knowledge graphs that couple abstract logic with real-world conditions.

   Example:
   [Agent] → [Action] ::mod(time="Past", mood="Assertion", value="Good")
   Here, Past, Assertion, and Good are attribute values originating from temporal, mood, and value Anchors, respectively.
   They constrain the contextual setting and epistemic attitude of the knowledge statement.

3. Cognitive Modulation.

Derived Anchors provide cognitive modulation cues for intelligent models (e.g., LLMs), assisting them in interpreting non-explicit factors such as tense, emotion, and intent, thereby improving semantic alignment and contextual coherence.

3.1.5 Classification of Derived Anchors

Category	Description	Examples
Temporal Anchors	Represent temporal points, intervals, or patterns	[Past], [Present], [Future], [Morning]
Spatial Anchors	Represent spatial location or positional relations	[Here], [There], [Australia], [Coordinate_XYZ]
Affective Anchors	Represent emotional or psychological states	[Joy], [Fear], [Anger], [Empathy]
Value Anchors	Represent moral, ethical, or systemic value orientations	[Good], [Evil], [Profit], [Fairness]
Modal Anchors	Represent possibility, necessity, or conditionality	[Possible], [Necessary], [Hypothetical], [Conditional]
Quantitative Anchors	Represent numerical magnitude or proportion	[One], [Many], [50%], [Majority]
Mood Anchors	Represent attitude or speech-act mood	[Assertion], [Question], [Request], [Doubt]
Perspective Anchors	Represent viewpoint or subjective stance	[FirstPerson], [ThirdPerson], [Objective], [Subjective]
State Anchors	Represent stages or phases within a process	[Begin], [Transition], [End], [Stable]

3.1.6 Summary

The Anchor System of STL consists of two complementary layers:

Core Anchors, which define the fundamental categories of semantic existence, and Derived Anchors, which define the dimensional fields in which semantic tension arises.

Together they form the semantic foundation of STL—a framework theoretically capable of mapping the full range of meanings expressible in natural language, while maintaining an extensible and open boundary for future cognitive and computational applications.

3.2 Path Expression

3.2.1 Definition

A Path Expression in the Semantic Tension Language (STL) represents the semantic channel that connects two Anchors.

It encodes the directional manifestation of meaning—how knowledge flows, maps, or transforms between semantic nodes within a structure.

Rather than merely denoting “relations between concepts,” a Path Expression models the manifestation process through which knowledge becomes explicit between Anchors.

STL expresses this process using a concise arrow syntax:

Example:
[Cause] ⟶ [Effect]
[Agent] ⟶ [Action]
[Concept] ⟶ [Definition]

The arrow indicates the direction of semantic tension transfer (source ⟶ target).

This directional encoding renders the relation both visualizable and computable, turning abstract semantics into an operational structure.

3.2.2 Core Functions of Path Expressions

Path Expressions serve three central functions within the STL semantic framework:

1. Semantic Projection – Defines how the meaning of one Anchor manifests in another.

   Example:
   [Concept_Gravity] ⟶ [Phenomenon_Falling]

   Here, the concept Gravity manifests as the phenomenon Falling, demonstrating semantic realization in a dependent node.

2. Structural Linking – Connects multiple Anchors into an integrated knowledge framework, enabling traceability, composition, and reasoning.

   Example:
   [Agent] ⟶ [Action] ⟶ [Result]

   This chain represents a structural mapping where an agent performs an action that leads to an outcome.

3. Directional Intent – Specifies the cognitive or logical projection direction: from concept to instance, from question to answer, or from subject to object. Directionality defines the logical coherence of the STL network and clarifies the flow of semantic causality.

3.2.3 Typology of Path Expressions

Depending on their semantic function and structural characteristics, STL identifies several primary types of Path Expressions:

Type	Description	Example
Semantic Path	Defines abstract or definitional relations between concepts	[Concept_A] → [Concept_B]

Type	Description	Example
Action Path	Describes the relation between an agent and its action	[Agent] → [Action]
Cognitive Path	Represents perception or understanding of a phenomenon by a cognitive subject	[Observer] → [Phenomenon]
Causal Path	Expresses causal or conditional mechanisms	[Cause] → [Effect]
Inferential Path	Represents logical or empirical reasoning processes	[Premise] → [Conclusion]
Reflexive Path	Denotes self-reference or structural closure	[Self] → [Self]

These categories are not mutually exclusive; in complex reasoning networks, multiple path types may co-occur or interlink within a single manifestation chain.

3.2.4 Path Composition and Manifestation Chains

In practical representation, multiple Path Expressions can be sequentially composed to form Manifestation Chains, which describe the generative or evolutionary process of knowledge:

Example:
[Concept] ⟶ [Event] ⟶ [Outcome] ⟶ [Value]

This chain illustrates how a concept becomes manifest in an event, the event produces an outcome, and the outcome is evaluated in terms of value.

Such chaining allows STL to model the propagation of meaning and its transformation across contexts, creating layered Semantic Path Networks.

These hierarchical structures provide a basis for knowledge tracing, semantic provenance, and reasoning.

3.2.5 Tension and Directionality

Every Path Expression embodies a semantic tension vector—the directional dependence and projection of meaning between Anchors.

Tension refers to the gradient of semantic force that drives meaning from one node toward another, forming the dynamic structure of a knowledge graph.

When multiple paths intersect or close upon themselves, they produce a tension network, a potential mechanism for representing complex semantics and higher-order logical relations.

Example:
[Question] → [Answer]
[Answer] → [Verifier]
[Verifier] → [Question]

Together these three paths form a semantic loop representing the full cognitive cycle of inquiry → response → verification.

Such reflexive configurations allow STL to express not only declarative knowledge but also epistemic processes and feedback structures.

3.2.6 Summary

Path Expression serves as the core dynamic mechanism of STL, transforming Anchors from static semantic units into interactive relations.

Through the explicit encoding of semantic directionality and tension, STL can represent the evolution of knowledge, inference chains, and cognitive cycles within a unified structural syntax.

In the next section (§ 3.3), we extend this mechanism with the Modifier Layer (::mod), which enriches Anchors and Paths with contextual attributes, enabling fine-grained semantic representation and situational reasoning.

3.3 Modifier Layer (::mod)

3.3.1 Definition

The Modifier in the Semantic Tension Language (STL) is a structural mechanism for encoding additional semantic information that enriches Anchors and Path Expressions.

It is attached to either a Path or an Anchor using the syntax ::mod(), specifying contextual attributes such as condition, state, intensity, or modality.

If Anchors define semantic entities and Paths define semantic relations, then the Modifier defines how these relations manifest—the manner, context, or attitude of semantic realization.

Through modifiers, STL unites computable semantics (formal logic) with perceptive semantics (contextual nuance), allowing time, mood, emotion, and belief strength to be expressed without breaking structural formality.

3.3.2 Syntax Form

The standard syntax of a Modifier is:

```
[Anchor_A] → [Anchor_B] ::mod(key1=value1, key2=value2, ...)
```

- ::mod denotes the modifier layer.
- Each key=value pair specifies an attribute and its corresponding value.
- Multiple attributes may appear, separated by commas.
- The value may reference attributes originating from Derived Anchors but never contain Anchors themselves, thereby preserving syntactic purity (see § 3.1.4).

Example
[Agent] → [Action] ::mod(time="Past", mood="Assertion", value="Good")

This statement expresses: “The agent performed a positive action in the past.”

Here time, mood, and value encode temporal, attitudinal, and evaluative dimensions of the same semantic structure.

3.3.3 Core Functions

Modifiers operate as the semantic modulation layer in STL and perform three essential functions:

1. Semantic Specification

They enable precise contextual description at the path level.

```
[Event] → [Outcome] ::mod(time="Future")
```

indicates that the event refers to a future prediction rather than an accomplished fact.

2. Structural Modulation

Modifiers can attach to either Anchors or Paths, forming layered semantic modulation structures.

```
[Belief] ⟶ [Action]
  ::mod(certainty=0.8)
  ::mod(emotion="Hope")
```

This means “A belief, with confidence 0.8, evokes an action infused with hope.”

Each layer refines meaning without altering structural connectivity.

3. Semantic Generalization and Abstraction

By combining multiple modifiers, STL can abstract semantic patterns across contexts.

```
[Agent] ⟶ [Goal] ::mod(time="Future", value="Good")
[Agent] ⟶ [Goal] ::mod(time="Past",  value="Bad")
```

The structural form remains constant while attribute values differ, enabling the system to compare and reason over contrasting semantic states.

3.3.4 Typology of Modifiers

According to their semantic role, STL defines several principal categories of modifiers:

Category	Function	Example
Temporal Mod	Expresses tense or temporal condition	::mod(time="Past")
Spatial Mod	Indicates spatial location or domain	::mod(location="Melbourne")
Affective Mod	Represents emotional or psychological state	::mod(emotion="Fear")
Logical Mod	Encodes logical rules, certainty, or causality	::mod(rule="causal", certainty=0.9)
Mood Mod	Marks grammatical or attitudinal mood	::mod(mood="Question")
Value Mod	Conveys moral, cultural, or evaluative stance	::mod(value="Good")
Cognitive Mod	Indicates cognitive intent or focus	::mod(intent="Explain", focus="Reason")
Causal Mod	Specifies conditional or causal relations	::mod(cause="A", effect="B")

Category	Function	Example
Quantitative Mod	Expresses magnitude, probability, or scale	::mod(probability=0.7, scale="High")

Each category corresponds to a dimension of semantic tension that modulates how information is interpreted within a knowledge graph.

3.3.5 Hierarchical Structure of Modifiers

Modifiers may be layered to express multi-dimensional semantic tension:

```
[Action] → [Result]
  ::mod(time="Present")
  ::mod(value="Neutral")
  ::mod(confidence=0.85)
```

Such stacking enables STL to represent multiple contextual dimensions simultaneously—temporal, evaluative, evidential—thereby supporting semantic dimension superposition.

This structure provides flexibility for reasoning engines to parse or weigh contextual layers according to task-specific priorities.

3.3.6 Functional Extension

At an advanced level, modifiers can evolve into semantic functions, allowing quantitative computation or logical comparison among contextual attributes:

```
::mod(time=Δt, confidence=f(P(evidence)))
```

Here, f(P(evidence)) defines a function over probabilistic evidence, suggesting that STL may serve as a semantic computation language.

This formulation establishes a foundation for interpretable reasoning engines and self-explanatory AI systems capable of operating on both structural and contextual semantics.

3.3.7 Relationship with Causal Paths

Causal Paths and Causal Modifiers represent two complementary aspects of the same phenomenon:

the former describes the structural existence of causality ([Cause] → [Effect]),

while the latter encodes the attribute tension that quantifies or contextualizes this relationship (::mod(cause="A", effect="B")).

Together they form a dual-layer model of causal manifestation—one structural, one attitudinal—enabling STL to express both logical form and semantic granularity.

3.3.8 Summary

The Modifier Layer is the key component that links semantic logic with semantic perception in STL.

It endows Anchors and Paths with contextual, emotional, evaluative, and temporal attributes, allowing STL to express the complexity and subtlety of natural language while preserving computational tractability.

Through the Modifier mechanism, STL accomplishes the transition from semantic structure to semantic behavior, forming a foundation for future research in knowledge modeling, semantic reasoning, and cognitively aligned AI.

3.4 Summary and Integration

The three core constructs of the Semantic Tension Language (STL)—Anchor, Path, and Modifier—jointly define the minimal compositional unit of structured meaning.

Anchors serve as semantic reference points representing entities, concepts, or cognitive roles;

Paths express directional relations and manifestation processes among these Anchors;

Modifiers provide contextual modulation that encodes time, value, mood, or causal tension.

Together, they transform semantic representation from static description into an active, tension-based structure that captures both the stability of knowledge and the dynamics of interpretation.

Within this framework, knowledge can be expressed as a connected graph in which each statement is traceable, interpretable, and computationally operable.

The Anchor–Path–Modifier triad thus functions as a semantic skeleton with a contextual membrane, enabling STL to unify logical precision with semantic richness.

These components also define the smallest implementable schema for future tooling: every valid STL statement can be decomposed into these three layers and serialized into interoperable formats for reasoning or data exchange.

The next section outlines how these theoretical constructs can be instantiated in practice—defining minimal compliance units, writing conventions, and potential mappings to machine-readable standards such as JSON and RDF.

4 – Implementation Outlook

The implementation of the Semantic Tension Language (STL) can be viewed as an evolutionary step beyond traditional symbolic representation frameworks such as RDF (Resource Description Framework) and OWL (Web Ontology Language). [14] [15] [16]

While RDF/OWL were designed for the Web era—to enable structured data to be machine-understandable through deterministic logical rules—their design philosophy remains fundamentally top-down.

They first define a rigid ontological schema, after which data must conform to that schema to be interpretable.

This approach ensures logical consistency but limits semantic adaptability: when a new domain or entity type emerges, a new ontology must be constructed before reasoning can proceed.

As a result, symbolic reasoning systems struggle with semantic transfer and structural analogy, both of which are essential for open-world and context-dependent reasoning.

STL proposes a complementary, bottom-up paradigm.

Instead of forcing knowledge to fit a pre-defined ontology, it represents meaning as a network of Anchors, Paths, and Modifiers—structures that emerge naturally from semantic relations rather than from fixed symbolic categories.

This design allows STL to handle “imperfect,” ambiguous, or overlapping data that better reflects the complexity of real-world semantics.

By encoding both structure and semantic tension, STL enables a flexible mapping between symbolic precision and linguistic fluidity, bridging the gap between rule-based and context-based reasoning systems.

From an implementation perspective, STL can be serialized into machine-readable formats such as JSON or RDF-like triples, forming a foundation for integration with existing knowledge-graph infrastructures.

A minimal STL-compliant statement can be expressed as:

```
{
 "anchor_from": "Agent",
 "anchor_to": "Action",
 "path_type": "Causal",
 "mod": {
  "time": "Past",
  "value": "Good"
 }
}
```

This structure preserves both the logical traceability of RDF triples and the semantic richness of natural language, offering a new interoperability layer for LLM-driven systems.

Future work will extend this mapping to full RDF/OWL compatibility and develop reasoning engines capable of operating on STL graphs while maintaining interpretability and traceable inference chains.

5 – Evaluation & Discussion

5.1 Feasibility Demonstration

However, practical deployment requires adherence to a standardized mapping protocol to ensure structural consistency and prevent semantic drift.

The STL framework already defines such a protocol layer, formalized in the STL Specification published at github.com/scos-lab/semantic-tension-language

.

This specification establishes the minimal compliance requirements for natural-language → STL translation, including attribute normalization, anchor-type resolution, and modifier encoding.

By adopting this protocol, STL-based systems can achieve reproducible and interoperable semantic mappings across different implementations, ensuring that each transformation follows a consistent and verifiable structure.

5.2 Interpretability and Cross-Domain Potential

One of the central motivations for developing the Semantic Tension Language (STL) is to enable interpretable reasoning in domains where knowledge is expressed through qualitative, context-dependent relations rather than quantitative rules.

To illustrate this capability, we present a case study drawn from Traditional Chinese Medicine (TCM)—a knowledge system that inherently encodes semantic tension between physical conditions, symbolic attributes, and therapeutic actions.

The example demonstrates how STL formalizes such reasoning into a transparent, machine-interpretable structure while preserving domain semantics.

Example: Encoding Traditional Chinese Medical Logic in STL

Example constructed by the author based on publicly available knowledge in Traditional Chinese Medicine; no copyrighted or third-party textual sources were used.

In TCM, the Five Flavors (五味)—sour (酸), bitter (苦), sweet (甘), pungent (辛), and salty (咸)—describe the functional tendencies of substances and their influence on the body.

Each flavor corresponds to a specific physiological effect (e.g., bitterness drains and dries, blandness percolates and eliminates dampness, sourness constrains and stabilizes).

Using STL, this hierarchical relation can be encoded as:

```
[五味] →[酸味] ::mod(role=subtype)
[五味] →[苦味] ::mod(role=subtype)
[五味] →[甘味] ::mod(role=subtype)
```

[五味] →[辛味] ::mod(role=subtype)
[五味] →[咸味] ::mod(role=subtype)

When addressing the syndrome of internal dampness (湿淫于内), the therapeutic reasoning—traditionally termed "treatment principle (立法)" and "flavor selection (选味)"—can be formalized as follows:

[湿淫] →[苦味] ::mod(role=main, principle="燥湿除浊")
[湿淫] →[热性] ::mod(role=main, principle="助苦以燥")
[湿淫] →[酸味] ::mod(role=assist, function="敛护")
[湿淫] →[淡味] ::mod(role=assist, function="渗泄利湿")
[苦味] →[泻燥坚] ::mod(type=mechanism)
[淡味] →[利窍渗泄] ::mod(type=mechanism)
[酸味] →[涩收] ::mod(type=mechanism)

Step 1 — STL Encoding

The first stage translates the natural-language description of the therapeutic principle into an STL graph.

Here, each anchor (*e.g., [湿淫], [苦味], [酸味]*) represents a semantic entity, while each path expresses a directional relation that embodies the treatment logic.

Modifiers (::mod) encode contextual attributes such as role (main vs. assist), principle ("dry dampness and eliminate turbidity"), and mechanistic functions.

This translation was performed according to the STL protocol publicly released at https://github.com/scos-lab/semantic-tension-language

, ensuring consistency and reproducibility.

Step 2 — LLM Reasoning Based on STL

The above STL structure was then provided as input to a large-language-model reasoning module.

When queried with:

"How should the treatment principle (立法) and Five-Flavor selection (选味) be established for internal dampness (湿淫于内)?"

the LLM interpreted the graph as a hierarchical semantic map, producing the following reasoning chain:

Main Directive — Derived from *[湿淫] →[苦味]*, stating the principle "燥湿除浊" (dry dampness and eliminate turbidity).

Reinforcing Mechanism — *[湿淫] →[热性]* indicates "助苦以燥" (enhancing bitterness through heat to reinforce drying).

Auxiliary Support — *[湿淫] →[酸味]* and *[湿淫] →[淡味]* define complementary roles: the sour flavor protects vitality through constriction, while the bland flavor facilitates moisture elimination.

Mechanistic Clarity — Secondary relations explain physiological actions:

[苦味] →[泻燥坚] (bitter flavor drains and solidifies),

[淡味] →[利窍渗泄] (bland flavor percolates and discharges moisture),

[酸味] →[涩收] (sour flavor constrains and stabilizes).

Each relation is explicitly traceable, providing a transparent path from clinical reasoning to computational representation.

Step 3 — Interpretability Outcome

Through STL's structural representation, a traditionally intuitive TCM concept is rendered into a precise semantic graph.

The reasoning process is no longer an opaque textual description but a network of anchors and paths with verifiable relations.

This approach illustrates STL's capacity to serve as a semantic bridge between symbolic and connectionist intelligence — linking the interpretability of structured knowledge with the adaptive reasoning of LLMs.

Moreover, it demonstrates cross-domain applicability: the same method can be extended to law, ethics, and social decision-making, where conceptual relations require both precision and contextual depth.

Summary

This experiment shows that STL enables reasoning systems to encode complex, non-numeric knowledge in a traceable, structurally interpretable form.

By binding semantic relations to explicit anchors and paths, STL transforms qualitative judgment into computable structure without sacrificing contextual richness.

Such integration suggests that STL could become a universal intermediate representation for explainable AI across diverse knowledge systems.

5.3 Limitations and Future Directions

Although the current evaluation demonstrates STL's interpretability and cross-domain potential, several limitations remain.

At present, the translation from natural language to STL is semi-automated, relying on large language models guided by the public STL Protocol.

While this ensures structural consistency, systematic quantitative evaluation of mapping precision, contextual fidelity, and inter-model reproducibility has not yet been established.

Future work will focus on expanding STL from a representational framework into a fully operational semantic infrastructure. This includes constructing benchmark datasets and quantitative metrics to evaluate anchor detection, modifier coherence, and path interpretability; extending STL serialization toward RDF and JSON-LD to ensure compatibility with existing Semantic Web frameworks and reasoning engines; and exploring STL's potential as an intermediate layer between symbolic reasoning and neural models, forming a structurally explainable interface between human-readable logic and adaptive machine semantics.

By advancing these directions, STL may evolve from a representational protocol into a general infrastructure for interpretable, cross-domain semantic computation.

6. Conclusion & Future Work

The Semantic Tension Language (STL) has been introduced as a new paradigm for representing knowledge in a manner that unifies structural precision with semantic adaptability.

By establishing a triadic framework composed of Anchors, Paths, and Modifiers, STL captures not only what is known but also how knowledge manifests and under what contextual conditions it holds.

This design transforms traditional representations of information from static symbolic forms into dynamic, tension-based structures capable of expressing ambiguity, multiplicity, and interpretive depth.

Through theoretical analysis and preliminary implementation, this study demonstrates that STL can operate as an effective intermediary layer between symbolic reasoning and neural understanding.

It provides a formal yet flexible means of encoding meaning that preserves interpretability while accommodating the fluid semantics of natural language.

In doing so, STL bridges a critical gap between the rigid determinism of classical logic and the probabilistic uncertainty of modern AI models, offering a unified structure for human–machine knowledge exchange.

The present work establishes the conceptual foundation of STL and its essential syntax, including how semantic entities, relations, and modifiers interact to form computable statements.

Future work will focus on formalizing standardized mapping protocols, refining interoperability with existing data frameworks such as RDF, and expanding STL's application to large-scale reasoning environments.

Ultimately, STL aims to serve as a universal language layer—one that preserves the traceability of knowledge while enabling systems to reason with the flexibility inherent in natural cognition.

Bibliography

[1] C. E. Shannon, "A Mathematical Theory of Communication.," Bell System Technical Journal, p. 27 (3), 1948.

[2] J. Gleick, The Information: A History, a Theory, a Flood, Pantheon Books, 2011.

[3] G. Frege, Begriffsschrift: A Formula Language, Modeled upon that of Arithmetic, 1879.

[4] Whitehead, A. N., Russell, B., Principia Mathematica, Cambridge University Press, 1910.

[5] A. Tarski, "The Concept of Truth in Formalized Languages," Oxford, Clarendon Press, 1936, pp. pp. 152--278 .

[6] N. Chomsky, Syntactic Structures, Mouton, 1957.

[7] J. Pearl, Causality: Models, Reasoning, and Inference, Cambridge University Press, 2009.

[8] A. d. G. S. B. H. B. P. D. P. H. K.-U. K. L. C. L. D. L. P. M. V. L. L. d. P. G. P. H. P. G. Z. Tarek R. Besold, Neural-Symbolic Learning and Reasoning: A Survey and Interpretation, arXiv:1711.03902 [cs.AI].

[9] M. R. Quillian, "Semantic Memory," in Semantic Information Processing. , MIT Press, 1968, p. pp. 227.

[10] C. J. F. a. J. B. L. Collin F. Baker, "The Berkeley FrameNet Project," in The 17th International Conference on Computational Linguistics, 1998.

[11] P. K. a. M. Palmer, "From TreeBank to PropBank," in In Proceedings of the Third International Conference on Language Resources and Evaluation (LREC'02), Las Palmas, Canary Islands – Spain, 2002.

[12] L. e. a. Banarescu, "Abstract Meaning Representation for Sembanking," in In Proceedings of the 7th Linguistic Annotation Workshop and Interoperability with Discourse, 2013.

[13] O. A. a. A. Rappoport, "The State of the Art in Semantic Representation," in In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Vancouver, Canada, 2017.

[14] T. H. J. &. L. O. Berners-Lee, "The Semantic Web," Scientific American, vol. 284(5), p. 34–43, 2001.

[15] D. &. G. R. Brickley, "RDF Schema 1.1," W3C Recommendation, 2014. [Online]. Available: https://www.w3.org/TR/rdf-schema/.

[16] W. O. W. Group, "OWL 2 Web Ontology Language Document Overview," W3C Recommendation, 2012. [Online]. Available: https://www.w3.org/TR/owl2-overview/.

[17] H. &. K. D. Knublauch, "Shapes Constraint Language (SHACL)," W3C Recommendation, 2017. [Online]. Available: https://www.w3.org/TR/shacl/.

[18] L. &. G. P. Moreau, "PROV-O: The PROV Ontology," W3C Recommendation, 2013. [Online]. Available: https://www.w3.org/TR/prov-o/.

[19] A. B. E. e. a. Hogan, "Knowledge Graphs," ACM Computing Surveys, vol. 54(4), p. 1–37.

[20] A. &. S. H. A. Newell, "Computer Science as Empirical Inquiry: Symbols and Search," Communications of the ACM, vol. 19, p. 113–126, 1976.

[21] J. &. H. P. J. McCarthy, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," Machine Intelligence, vol. 4, p. 463–502, 1969.

[22] D. E. &. M. J. L. Rumelhart, Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MIT Press, 1986.

[23] G. E. Hinton, "Mapping part-whole hierarchies into connectionist networks," Artificial Intelligence, Vols. 46, 1–2, p. 47–75, 1990.

[24] P. Smolensky, "On the Proper Treatment of Connectionism," BEHAVIORAL AND BRAIN SCIENCES, vol. 11, pp. 1-74, 1988.

[25] G. Marcus, "The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence," arXiv preprint arXiv:2002.06177, 2020.