# Semantic Tokens v1.0:
# A Foundational Meaning Layer for Deterministic AI Systems

Robert Hansen
Design Logic
robert.h.designlogic@gmail.com

November 18, 2025

**Abstract**

Modern AI systems rely on surface-form text tokens that blend meaning with raw language, producing drift, ambiguity, and instability in long-context reasoning. This often leads to hallucination, misalignment, and unreliable multi-step behavior, especially in distributed or multi-agent settings.

Semantic Tokens (STs) introduce a governed, typed, meaning-level representation that preserves intent, structure, and relational semantics independently of text. This paper defines the Semantic Token standard, formalizes the token schema, introduces an execution model, and outlines how STs enable deterministic reasoning and multi-agent coordination. By stabilizing meaning over time, STs support long-horizon planning, safer alignment, and cross-tool interoperability. Semantic Tokens serve as a substrate for reliable AI orchestration at scale, bridging probabilistic language models with structured, deterministic computation.

## 1 Introduction

Large language models (LLMs) operate on sequences of subword text tokens. These tokens encode surface form (spelling, punctuation, morphology), but not explicit semantic intent. As a result, the model learns statistical associations over string fragments rather than governed units of meaning.

This design is extremely effective for open-ended generation, but it introduces structural weaknesses when systems are used for:

- multi-step reasoning and planning,
- multi-agent coordination,
- tool or API orchestration,
- long-horizon interaction with humans or other agents.

In these settings, *meaning* must remain stable across turns, agents, and tools. Text tokens alone do not provide that stability.

We propose **Semantic Tokens (STs)**: a standard for typed, governed, meaning-level units that can be shared across agents and execution environments. STs do not replace text tokens; they complement them with an explicit semantic substrate.

## 2 Problem: Text Tokens Are Not Enough

LLMs interpret meaning probabilistically: given a context window of text tokens, the model predicts the next token distribution. This implicitly entangles semantics with surface form and leads to several well-known failure modes:

- **Conceptual drift:** Over long interactions, concepts gradually shift or collapse, especially when compressed or summarized repeatedly.
- **Unstable chains of thought:** Small perturbations in wording or order can cause qualitatively different reasoning paths.
- **Agent divergence:** Two agents consuming the same textual summary may develop different internal representations and plans.
- **Hallucination under ambiguity:** When semantics are under-specified, models fill in gaps without a clear record of assumptions.
- **Intent misalignment:** Changes in phrasing can mutate the inferred objective, even when the underlying goal has not changed.

In distributed AI systems, these weaknesses compound. When agents pass only text, there is no explicit representation of:

- what entities and relations are involved,
- which constraints must be respected,
- what intent the message carries (inform, propose, commit, request, etc.),
- which parts of the content are stable commitments vs. speculative hypotheses.

Text-based protocols can approximate this via schemas, ontologies, or conventions, but they remain fragile: the *semantics* is implicit in how humans and models interpret those strings. We argue that distributed AI requires a first-class meaning layer.

# 3 Semantic Token Model

Semantic Tokens provide such a layer. Intuitively, an ST is a *typed, atomic unit of meaning* that can be serialized, stored, and manipulated independently of any particular textual phrasing.

## 3.1 Overview

Conceptually, each message in a system carries two parallel views:

1. **Surface view:** free-form natural language, optimized for human readability and LLM fluency.
2. **Semantic view:** a set of Semantic Tokens representing intent, entities, relations, states, and constraints.

The semantic view is governed by a schema that defines:

- token types and fields,
- invariants and constraints,
- relationships between tokens,
- execution-time expectations (e.g., which tokens are required for a valid plan step).

## 3.2 Typed Meaning Units

At minimum, the Semantic Token layer distinguishes the following families of tokens:

- **Intent tokens** (e.g., `REQUEST_ACTION`, `PROPOSE_PLAN`, `REPORT_STATE`).
- **Entity tokens** (objects, agents, resources, tasks) with stable identifiers.
- **Relation tokens** capturing links such as ownership, dependency, precedence, or containment.
- **State tokens** representing conditions, flags, or environment variables.

- **Constraint tokens** specifying hard or soft requirements (deadlines, budgets, safety rules).

Tokens are *not* tied to strings like "river" or "city". Instead, they refer to canonical concepts that may be rendered into many surface forms.

### 3.3 Schema-Bound Representation

Each token instance is governed by a schema. At a high level, we can view an ST as:

$$\text{ST} := (\texttt{type}, \texttt{id}, \texttt{payload}, \texttt{constraints}, \texttt{metadata}) \tag{1}$$

where:

- `type` is a member of a finite, versioned token ontology,
- `id` is a stable reference for this semantic object,
- `payload` contains type-specific fields,
- `constraints` encode invariants or obligations,
- `metadata` attaches provenance, timestamps, confidence, and links.

By fixing a schema, we obtain a representation that is:

- **machine-checkable** (Binder-style validation),
- **model-agnostic** (independent of any particular LLM),
- **serializable** into JSON, protobuf, or other wire formats,
- **versioned** for long-horizon compatibility.

## 4 Semantic Token Schema

This section sketches a minimal but practical schema suitable for multi-agent coordination. In production, this schema would be extended and versioned.

### 4.1 Core Fields

We define a top-level JSON shape for a token instance:

```
{
  "token_id": "ST-000123",
  "token_type": "INTENT.REQUEST_ACTION",
  "scope": "AGENT_SESSION",
  "payload": { ... },
  "constraints": { ... },
  "metadata": {
    "created_at": "2025-11-17T03:10:00Z",
    "created_by": "agent://planner_1",
    "confidence": 0.94,
    "provenance": ["msg://1234"]
  }
}
```

The `payload` structure is determined by `token_type`. For example, an intent token might contain:

```
"payload": {
  "action": "GENERATE_SPEC",
  "target_entity": "ENT-API_DESIGN_42",
  "urgency": "HIGH"
}
```

### 4.2 Minimal Coordination Ontology

As a starting point, we can define the following token types for agent coordination:

- `INTENT.REQUEST_ACTION`
- `INTENT.PROPOSE_PLAN`
- `INTENT.CONFIRM`
- `ENTITY.AGENT`
- `ENTITY.RESOURCE`
- `RELATION.DEPENDS_ON`
- `STATE.ENV_VARIABLE`
- `CONSTRAINT.SAFETY_RULE`

Crucially, these types are globally understood within the system. Agents can extend the ontology locally, but shared behavior depends on the common subset.

### 4.3 Constraints and Invariants

Constraints can be attached both at the schema level and at the instance level. Examples include:

- field cardinality (e.g., every `REQUEST_ACTION` must reference at least one `ENTITY`),
- type compatibility (e.g., `RELATION.DEPENDS_ON` must link valid entities),
- temporal constraints (e.g., deadlines, expiration times),
- safety invariants (e.g., actions forbidden under certain states).

A Binder-like component can validate tokens and reject messages that violate invariants, providing deterministic guardrails around probabilistic models.

## 5 Execution Model

The execution model specifies how STs are produced, consumed, and transformed in a running system. We assume a typical loop:

1. Human or agent emits natural language.
2. A *semantic injector* generates a set of candidate Semantic Tokens from the text.
3. Binder validates and normalizes the tokens according to the schema.
4. Agents plan and act over the semantic representation.
5. Text is generated *from* the updated semantic state when needed for humans or other systems.

### 5.1 Text → Semantic Tokens

Generation of STs from text can be implemented via:

- prompted LLMs constrained to output valid JSON shapes,
- rule-based extractors for known patterns (IDs, dates, amounts),
- hybrid approaches using both structure and model inference.

The critical requirement is that the output passes schema validation. When ambiguity is high, the system can attach lower confidence scores or explicit hypotheses rather than pretending to know the truth.

### 5.2 Semantic Tokens → LLM Context

When LLM calls are needed, semantic state can be re-encoded into structured prompts, e.g.:

```
System: You are operating on the following semantic state:
- Intent: REQUEST_ACTION (GENERATE_SPEC) on ENTITY API_DESIGN_42
- Constraints: SAFETY_RULE SR-7, deadline T+24h
- Relations: API_DESIGN_42 DEPENDS_ON SCHEMA_10

User: Produce the next concrete step that respects all constraints.
```

By grounding prompts in explicit STs, we reduce drift and keep meaning stable even as natural language descriptions vary.

### 5.3 Deterministic Behavior Over Time

Determinism here is not about bitwise identical outputs, but about *stable semantics*:

- Given the same semantic state and policies, the system produces equivalent decisions.
- Small perturbations in text do not silently mutate the underlying meaning.
- Agent coordination protocols can reason over explicit intent and constraints, not inferred guesswork.

Semantic Tokens make this possible by separating persistent meaning from transient phrasing.

## 6 Applications

### 6.1 Multi-Agent Coordination

In a multi-agent system, each message can carry both text and a bundle of Semantic Tokens. Agents can:

- negotiate over explicit intent tokens (*propose*, *accept*, *reject*),
- maintain shared views of entities and resources via stable IDs,
- detect and resolve conflicts via constraint tokens.

A minimal proof-of-concept can be built with two agents exchanging typed messages, comparing drift between a text-only protocol and an ST-augmented protocol.

### 6.2 Tool and API Integration

Tools can be bound to specific token types. For example:

- a database tool might be triggered by `INTENT.QUERY_DATA` tokens,
- a deployment tool might require `CONSTRAINT.SAFETY_RULE` tokens to be satisfied before executing.

Because tokens are model-agnostic, tools can operate on a stable schema even as models are upgraded or swapped.

### 6.3 Alignment and Auditability

STs provide a natural substrate for alignment:

- Intent is explicit and can be logged.
- Constraints and safety rules are first-class objects, not hidden in prompts.
- Decisions can be traced back to the semantic state that produced them.

This enables audit trails and post-hoc analysis of failures without reverse-engineering raw text transcripts.

## 7 Evaluation Plan

A full empirical evaluation is beyond the scope of this introductory paper, but we outline an initial plan.

### 7.1 Drift Measurement Experiment

1. Define a minimal Semantic Token schema for agent coordination.
2. Implement two agents:
   - **Baseline:** communicates via text-only messages.
   - **ST-augmented:** communicates using both text and ST bundles.
3. Design multi-turn tasks that require stable intent and shared state.
4. Measure:
   - task completion rate,
   - number of clarification turns,
   - semantic drift (changes in inferred goals or entities),
   - error types (hallucination, miscoordination, constraint violation).
5. Compare performance between the two conditions.

We hypothesize that ST-augmented agents will exhibit lower drift, fewer misalignments, and more reliable task completion under long-horizon interactions.

## 8 Related Work

Semantic Tokens intersect with several existing research threads:

- **Formal Semantics and AMR:** Abstract Meaning Representation and related formalisms provide graph-based semantic structures. STs share the graph idea but focus on operational, schema-bound tokens for distributed systems.
- **Knowledge Graphs and Ontologies:** Knowledge graphs encode entities and relations, but are often static or offline. STs represent online, transient meaning units that can flow through runtime systems.
- **Program Synthesis and DSLs:** Domain-specific languages can express structured operations, but typically require strict syntax. STs occupy a middle ground: structured like a DSL, but generated and consumed in tandem with natural language.
- **Agent Communication Languages:** Protocols such as FIPA ACL specify communicative acts (inform, request, etc.). STs can be seen as a modern, LLM-compatible realization of similar ideas with explicit typing, constraints, and integration into model prompts.

## 9 Discussion and Future Work

Semantic Tokens v1.0 is intentionally minimal. Many extensions are possible:

- richer ontologies for tasks, plans, and world models,
- probabilistic or fuzzy semantics attached to tokens,
- hierarchical tokens (macro-tokens composed of subgraphs),
- integration with blockchain-style ledgers for global knowledge graphs,
- standardized open-source schemas for common coordination patterns.

The central claim is not that any specific token ontology is final, but that *a governed semantic substrate is necessary* for deterministic AI orchestration at scale.

## 10 Conclusion

Text tokens made modern LLMs possible, but they are not sufficient for reliable, deterministic, multi-agent AI systems. By introducing Semantic Tokens as typed, schema-bound meaning units, we separate stable semantics from surface form and enable new capabilities:

- deterministic meaning preservation across turns and agents,
- cross-agent semantic stability,
- intent locking and constraint-aware planning,
- tool and API integration over a shared semantic substrate,
- alignment, auditability, and governance at the level of intent and state.

Semantic Tokens v1.0 is a starting point: a practical, implementable standard that invites experimentation, extension, and open collaboration. The authors invite feedback from researchers and practitioners working on agent orchestration, distributed AI protocols, and meaning-centered architectures.