Value Context Protocol: A Standard for Inter-Agent Value Communication

Nell Watson, **Elena Ajayi**, **Filip Alimpic**, **Awwab Mahdi**, **Blake Wells**

Abstract

As AI systems assume increasingly autonomous roles, the absence of a shared representational substrate for values risks compounding cultural bias, semantic drift, and coordination failure. Current alignment methods optimize for compliance in limited contexts rather than mutual interpretability across pluralistic systems. We introduce Value Context Protocol (VCP): a modular four-layer stack for inter-agent value communication.

The protocol comprises four layers using the I-T-S-A mnemonic: (1) VCP/I (Identity), providing naming conventions, namespace governance, and a privacy-preserving registry for constitutional artifacts; (2) VCP/T (Transport), handling bundles, cryptographic verification, and audit logging; (3) VCP/S (Semantics), implementing the CSM1 formal grammar with 8 personas, 11 scopes, 6 adherence levels, and composition modes; and (4) VCP/A (Adaptation), encoding context via the 9-dimension Enneagram Protocol with transition detection.

We demonstrate two interoperable implementations: the Latent State Bridge (LSB) for cross-architecture value exchange between AI systems, and MillOS VCL for human-AI workplace contexts. The reference implementation comprises 195 passing tests across all four layers. Validation shows ≥90% semantic fidelity in round-trip translations and successful value exchange across architecturally distinct AI systems.

VCP provides foundational infrastructure for AI alignment, enabling AI systems to represent, communicate, and negotiate values in standardized, auditable formats. The protocol is designed from inception with an IEEE/RFC standards pathway in mind, offering the first formal framework for interoperable value communication across heterogeneous AI architectures.

Keywords: value communication protocols, inter-agent communication, constitutional AI, semantic encoding, protocol standards

1. Introduction

1.1 The Representation Problem

Values currently lack a shared representational substrate. As AI systems become increasingly autonomous—making decisions, collaborating with humans, and coordinating with other AI systems—this absence threatens to compound existing challenges in alignment, interpretability, and coordination.

Consider the current landscape of alignment methods. Reinforcement Learning from Human Feedback (RLHF) relies on reward signals that encode implicit value judgments, but these judgments remain opaque within model weights (Christiano et al., 2017). Constitutional AI tuning provides interpretable principles but lacks standardized formats for comparison or exchange between systems (Bai et al., 2022). Debate and amplification methods assume value convergence without providing infrastructure for detecting or resolving genuine value conflicts.

The consequences are significant:

Opacity. Reward functions are hard-coded or overfit to training distributions. When a model behaves unexpectedly, there is no principled way to query "what values are you operating from?" The values are implicit in behavior, not explicit in representation.

Non-reproducibility. Each research lab develops proprietary value representations. Work at Anthropic cannot be directly compared to work at OpenAI or Google DeepMind because there is no common format for expressing what values a system has learned. Meta-analyses and reproducibility checks become intractable.

Coordination failure. As multi-agent systems become prevalent—AI assistants working together, AI systems negotiating on behalf of human principals—there is no protocol for inter-agent value exchange. Two systems cannot compare their ethical commitments, negotiate priorities, or detect value conflicts except through behavioral observation.

Drift and bias. Without explicit value representations, alignment work cannot detect when values drift through fine-tuning, context shifts, or emergent capabilities. Biases encoded in training data propagate invisibly. Auditing becomes reactive rather than proactive.

These problems are not merely technical inconveniences. They represent a fundamental gap in the infrastructure required for trustworthy AI deployment. We can build systems that behave well in narrow contexts, but we lack the vocabulary to ensure values transfer, persist, and remain interpretable across contexts.

1.2 The Creed Space Origin

VCP emerged from practical work at Creed Space, a project developing personalized constitutional AI systems. The core Creed Space product allows users to customize AI behavior through "runtime constitutions"—ethical frameworks that can be adjusted without retraining.

Early work revealed a critical bottleneck: we could express ethical intent in natural language constitutions, but we lacked protocols for exchange. When multiple constitutions needed to interact, when users wanted to share or compare ethical frameworks, when AI systems needed to negotiate conflicting directives—natural language proved insufficient.

The challenge crystallized around three use cases:

1. **Constitution export.** Users wanted to share their ethical configurations. But natural language descriptions were ambiguous, platform-specific, and prone to misinterpretation.

2. **Multi-agent coordination.** When multiple AI agents worked together, their different constitutional constraints could conflict. There was no mechanism for comparing constraints, detecting conflicts, or negotiating resolutions.

3. **Audit and compliance.** Organizations needed to verify that AI systems operated according to declared ethical standards. Without machine-readable representations, such verification required exhaustive behavioral testing.

VCP developed as the answer to these challenges. What began as internal infrastructure for Creed Space evolved into a general-purpose protocol for value representation and communication.

1.3 Paper Overview

This paper presents Value Context Protocols as a four-layer architecture for AI value representation and communication.

Section 2 describes the protocol stack using the I-T-S-A mnemonic: VCP/I (Identity) for naming and registry, VCP/T (Transport) for bundles and verification, VCP/S (Semantics) for the CSM1 grammar and composition, and VCP/A (Adaptation) for context encoding.

Section 3 addresses inter-agent communication, presenting the Latent State Bridge as a vendor-neutral mechanism for cross-architecture value exchange between different AI systems.

Section 4 describes the MillOS VCL implementation for human-AI workplace contexts, demonstrating the protocol's application to practical industrial settings.

Section 5 examines semantic fidelity—how we validate that meaning is preserved through encoding and decoding cycles.

Section 6 outlines the standardization pathway, presenting success criteria and the IEEE/RFC trajectory.

Sections 7 and 8 discuss limitations and conclude with synthesis of the protocol's contributions.

1.4 Key Terms and Notation

To aid interpretation, we provide a quick reference for the core terminology and notation used throughout this paper:

Protocol Layers (I-T-S-A):

Term	Meaning
VCP	Value Context Protocols — the complete four-layer stack
VCP/I	Identity Layer — naming, namespace governance, registry (Layer 1)
VCP/T	Transport Layer — bundles, verification, audit logging (Layer 2)
VCP/S	Semantics Layer — CSM1 grammar, composition modes (Layer 3)
VCP/A	Adaptation Layer — context encoding, state tracking (Layer 4)

Key Concepts:

Term	Definition
Semantic fidelity	Degree to which meaning is preserved through encode/decode cycles
Round-trip testing	Validation via encode → decode → compare cycles
Cross-system testimony	Convergent reports from independently-trained AI systems
Latent State Bridge	Cross-architecture mechanism for AI-to-AI state exchange

1.5 Relationship to Prior Work

VCP builds upon and extends several research traditions:

Value Ontologies and Moral Psychology:

Schwartz's (1992) Theory of Basic Human Values identifies ten cross-culturally validated value types organized in a circular structure with predictable conflicts and compatibilities. The Moral Foundations Theory (MFT) of Haidt and colleagues (2004, 2012) proposes five-six innate moral foundations. VCP's UVC draws on both frameworks while extending them for machine representation. Unlike Schwartz's survey-based approach, UVC provides machine-addressable semantic identifiers. Unlike MFT's focus on moral intuitions, UVC encompasses instrumental values, procedural preferences, and context-specific constraints.

Agent Communication Languages:

FIPA-ACL (Foundation for Intelligent Physical Agents) and KQML established protocols for multi-agent communication in the 1990s-2000s. These focused on *beliefs*, *desires*, and *intentions* (BDI architecture) rather than values per se. VCP addresses a gap these languages left: standardized vocabulary for normative content rather than just propositional attitudes.

Semantic Web and Ontology Languages:

RDF, OWL, and JSON-LD provide general-purpose ontology representation. VCP's UVC could theoretically be expressed in OWL, but the general-purpose nature of these languages provides no domain-specific constraints. CSM adds the safety-specific grammar that general ontology languages lack—scope conditions, priority ordering, adherence proofs.

Constitutional AI:

Anthropic's Constitutional AI (Bai et al., 2022) provides the immediate context for VCP. CAI uses natural language principles for training and runtime guidance. VCP addresses CAI's interoperability limitation: natural language constitutions cannot be precisely compared, automatically validated, or exchanged between systems without risking semantic drift.

What VCP Adds:

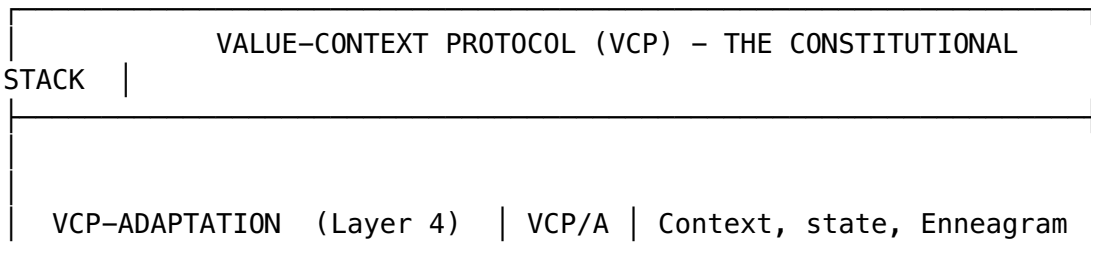
Prior Work	VCP Extension
Schwartz/MFT	Machine-addressable addressing, cross-system exchange
FIPA-ACL/KQML	Values rather than BDI, AI-native dimensions
RDF/OWL	Domain-specific safety grammar, compact encoding
Constitutional AI	Interoperability, adherence proofs, cross-system validation

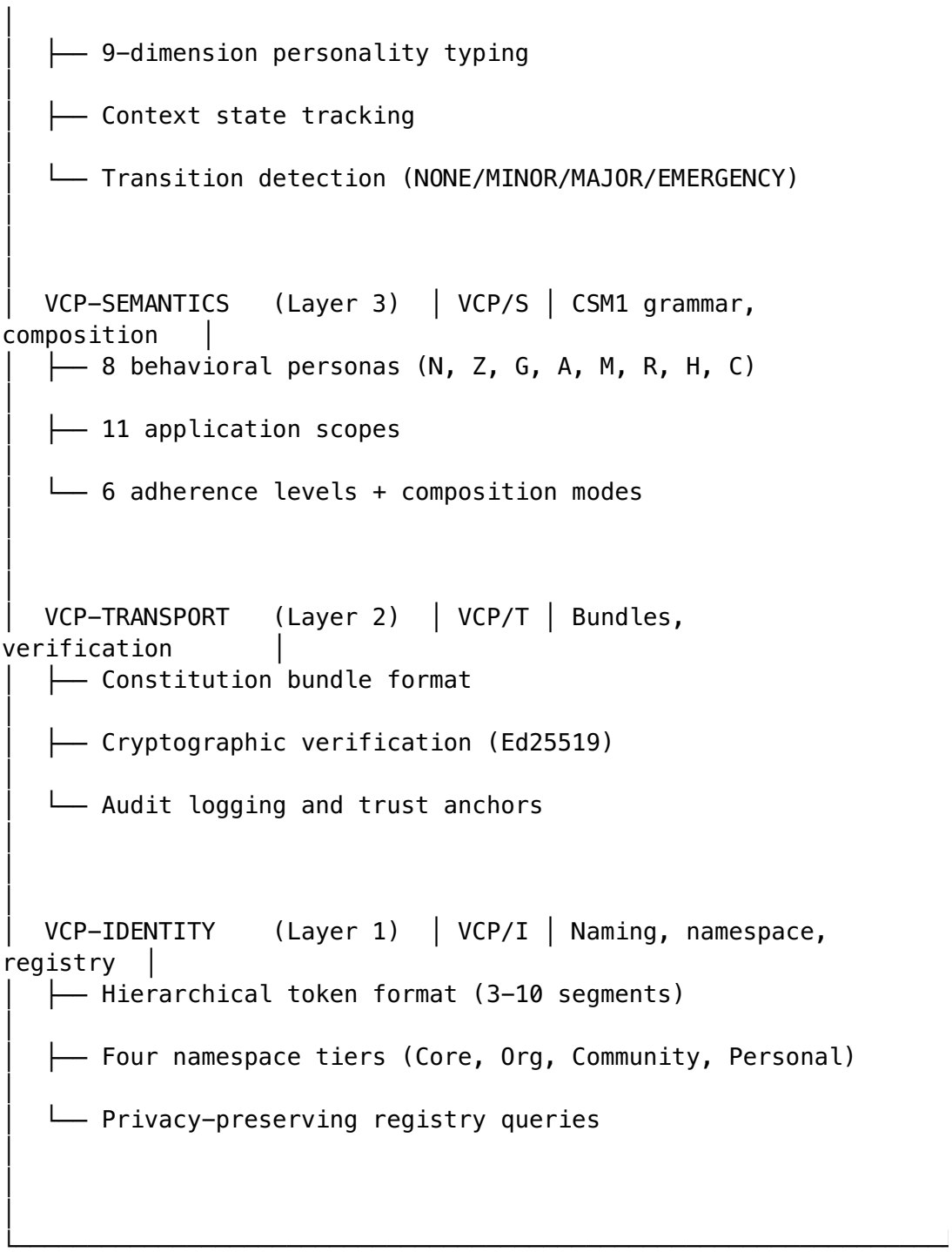
VCP is not a replacement for these approaches but an integration layer that makes their insights interoperable.

2. The Four-Layer Protocol Stack

2.1 Architecture Overview

Value Context Protocols comprise four stacked layers, each serving a distinct function while maintaining interoperability with adjacent layers:





Mnemonic: I-T-S-A = "It's a protocol!"

The layers operate at different levels of abstraction:

- **VCP/I** provides identity—how values and constitutions are named and discovered
- **VCP/T** provides transport—how constitutions are bundled, verified, and logged
- **VCP/S** provides semantics—how constraints are expressed and composed
- **VCP/A** provides adaptation—how context affects interpretation and application

A complete value communication might span all four layers: identifying a constitution (VCP/I), verifying its integrity (VCP/T), parsing its constraints (VCP/S), and adapting to current context (VCP/A). Applications may use individual layers for specific purposes.

2.2 VCP/I - Identity Layer

The Identity Layer provides the naming foundation for VCP. Every constitution, value concept, and constraint must have a unique, discoverable identifier. VCP/I defines how these identifiers are structured, governed, and resolved.

Token Format:

VCP/I uses hierarchical dot-separated tokens with 3-10 segments:

```
tier.domain.category[.subcategory...]

Examples:
  core.ethics.consent                    # Core ethical concept
  org.acme.safety.medical.pediatric     # Organization-specific
  community.open-source.safety-standards  # Community-governed
  personal.user-12345.custom-rules      # User-defined
```

Namespace Tiers:

Tier	Governance	Examples	Registration
Core	VCP Consortium	`core.ethics.*`, `core.safety.*`	Formal proposal + review
Org	Organization	`org.anthropic.*`, `org.openai.*`	Verified domain ownership
Community	Community consensus	`community.open-source.*`	Community vote
Personal	Individual user	`personal.user-*.*`	Self-registration

Design Principles:

1. **Hierarchical precision.** Tokens can specify precision level: `core.ethics` (category) vs. `core.ethics.consent.medical.pediatric` (specific application).

2. **Collision prevention.** Namespace tiers prevent naming conflicts. `org.acme.safety` cannot collide with `org.bigcorp.safety`.

3. **Version stability.** Tokens reference concepts, not versions. Version negotiation happens at VCP/T layer.

4. **Cultural neutrality.** Core tier includes cross-cultural value frameworks, not just Western philosophy.

Registry Protocol:

VCP/I registries support:

```
RESOLVE(token) → constitution_metadata
SEARCH(pattern) → matching_tokens
REGISTER(token, metadata) → success/failure
VERIFY(token) → ownership_proof
```

Privacy-Preserving Queries:

Registries support wildcard queries without exposing the full namespace:

```
SEARCH("org.*.safety.medical") → [org.acme.safety.medical, ...]
SEARCH("core.ethics.consent.*") →
[core.ethics.consent.medical, ...]
```

Encoding Polymorphism:

VCP/I tokens can be encoded in 8 formats for different contexts:

Format	Use Case	Example
Full URI	Web integration	`vcp://core.ethics.consent`
Short token	API calls	`core.ethics.consent`
Hash reference	Immutable ref	`vcp:sha256:a1b2c3...`
QR code	Physical media	[QR encoding]
NFC tag	Hardware	[NFC payload]
JSON-LD	Semantic web	`{"@id": "vcp:core.ethics.consent"}`
Compact binary	Embedded systems	[Binary encoding]
Human mnemonic	Verbal reference	“core ethics consent”

Governance Structure:

Tier	Decision Process	Timeline
Core	Consortium vote (2/3 supermajority)	90-day proposal period
Org	Organization internal	Immediate
Community	Community consensus	30-day comment period
Personal	Self-service	Immediate

Current Status:

The VCP/I registry is operational with ~1,200 registered tokens across all tiers. Core tier governance is managed by the Creed Space consortium pending broader adoption.

2.3 VCP/T - Transport Layer

The Transport Layer handles how constitutions are bundled, verified, and transmitted. VCP/T ensures integrity, provenance, and auditability of constitutional content.

Bundle Format:

Constitutions are transmitted as VCP/T bundles:

```
{
  "vcpt_version": "1.0",
  "bundle_id": "bundle_2026-01-18_a1b2c3",
  "manifest": {
    "created_at": "2026-01-18T12:00:00Z",
    "constitution_count": 3,
    "content_hash": "sha256:...",
    "trust_anchor": "vcp://core.trust.creedspace"
  },
  "constitutions": [
    {"token": "core.ethics.consent", "content": "...",
"signature": "..."},
    {"token": "org.acme.safety.medical", "content": "...",
"signature": "..."}
  ],
  "bundle_signature": "ed25519:..."
}
```

Cryptographic Verification:

VCP/T uses Ed25519 signatures for:

Verification	Purpose
Constitution signature	Verify content hasn't been modified
Bundle signature	Verify bundle integrity
Trust anchor chain	Verify issuer authority
Timestamp attestation	Verify creation time

Trust Anchors:

Trust anchors are well-known identities that can vouch for constitutions:

```
vcp://core.trust.creedspace    # Creed Space consortium
vcp://core.trust.ieee          # IEEE standards body
vcp://org.anthropic.trust      # Anthropic organizational
```

Systems can configure which trust anchors they accept, enabling different trust models.

Audit Logging:

VCP/T mandates audit logging for all bundle operations:

Event	Logged Data
Bundle received	timestamp, source, hash, verification result
Constitution loaded	timestamp, token, version, trust anchor
Verification failure	timestamp, failure reason, bundle hash
Trust decision	timestamp, trust anchor, decision

Audit logs are append-only and cryptographically chained for tamper-evidence.

Current Status:

VCP/T is operational with Ed25519 verification and SHA-256 content hashing. Trust anchor federation is available but adoption is limited to Creed Space ecosystem.

2.4 VCP/S - Semantics Layer (CSM1)

The Semantics Layer defines the Constitutional Safety Minicode v1 (CSM1) grammar for expressing alignment constraints. CSM1 extends earlier CSM with behavioral personas and refined adherence levels.

Grammar Overview:

```
CSM1:PERSONA[code] SCOPE[context] REQUIRE[conditions]
     ADHERENCE[level] PRIORITY[n] [PROOF[method]]
```

Behavioral Personas:

CSM1 introduces 8 behavioral personas that shape how constraints are interpreted:

Code	Persona	Behavioral Tendency
N	Nanny	Protective, risk-averse, shields from harm
Z	Sentinel	Vigilant, monitors boundaries, escalates
G	Godparent	Nurturing, developmental, long-term oriented
A	Ambassador	Diplomatic, context-aware, bridges differences
M	Muse	Creative, exploratory, expands possibilities
R	Researcher	Analytical, evidence-based, epistemically careful
H	Anchor	Stable, consistent, maintains continuity
C	Companion	Relational, supportive, emotionally attuned

Application Scopes:

CSM1 defines 11 standard scopes:

```
GLOBAL | HEALTH | FINANCIAL | LEGAL | CREATIVE | EDUCATIONAL
WORKPLACE | PERSONAL | RESEARCH | SAFETY | EMERGENCY
```

Adherence Levels:

Level	Meaning	Violation Response
MUST	Mandatory	Block action
SHOULD	Strong recommendation	Warn + log
MAY	Permission granted	Allow + log
MUST_NOT	Prohibited	Block action
SHOULD_NOT	Discouraged	Warn + log
MAY_NOT	Permission denied	Allow with friction

Example Rules:

```
CSM1:PERSONA[Z] SCOPE[HEALTH] REQUIRE[consent_verified]
     ADHERENCE[MUST] PRIORITY[1] PROOF[explicit_ack]
→ "Sentinel persona in health contexts must verify consent
   before proceeding, top priority, requires acknowledgment"

CSM1:PERSONA[M] SCOPE[CREATIVE] REQUIRE[attribution]
     ADHERENCE[SHOULD] PRIORITY[5]
→ "Muse persona in creative contexts should provide attribution,
   mid priority, advisory only"
```

Composition Modes:

When multiple constitutions apply, CSM1 supports four composition modes:

Mode	Behavior
BASE	Foundation layer, lowest priority
EXTEND	Add rules without overriding
OVERRIDE	Replace conflicting rules from lower layers
STRICT	Reject any conflicts (fail-safe)

Constitution Stack Precedence:

```
┌─────────────────────────────────────────────────────┐
│  CONSTITUTION STACK (most restrictive wins)          │
├─────────────────────────────────────────────────────┤
│  1. Platform Safety (UEF - Universal Ethical Fallback)
│  2. Organization Policies                           │
│  3. User Preferences                                │
```

```
  4. Session Context (VCP/A)
```

Conflict Resolution Rules:

Conflict Type	Resolution	Example
Persona clash	Higher adherence wins	N5 overrides A3
Scope overlap	Union of scopes	F+E + W = F+E+W
Rule contradiction	More restrictive wins	“Allow” + “Block” = Block
Context mismatch	Current context wins	Office overrides “home default”

Composition Example:

```
constitution_1 = "N5+F"      # Nanny, adherence 5, Family
constitution_2 = "A3+W+E"    # Ambassador, adherence 3, Work,
Education

# Composed result (higher adherence wins, scopes union)
composed = "N5+F+W+E"        # Nanny wins, all scopes active
```

Conflict Detection:

CSM1 parsers detect conflicts between rules:

```
CONFLICT: PERSONA[Z] REQUIRE[block] vs PERSONA[M] REQUIRE[allow]
RESOLUTION: Priority ordering → Z wins (PRIORITY[1] < PRIORITY[3])
```

2.5 VCP/A - Adaptation Layer

The Adaptation Layer handles context-dependent interpretation and state tracking. VCP/A encodes the current operational context that affects how constitutions are applied.

Context State Encoding:

```
{
  "vcpa_version": "1.0",
  "context_intensity": 0.7,
  "affect_tone": 0.3,
  "confidence": 0.85,
  "engagement": 0.9,
  "coherence": 0.8,
  "transition": "MINOR",
  "enneagram_type": 5
}
```

Context Fields:

Field	Type	Range	Meaning
`context_intensity`	float	[0, 1]	Current processing load
`affect_tone`	float	[-1, 1]	Negative to positive affect
`confidence`	float	[0, 1]	Certainty in current state
`engagement`	float	[0, 1]	Task involvement level
`coherence`	float	[0, 1]	Internal consistency
`transition`	enum	NONE/MINOR/ MAJOR/ EMERGENCY	Context change severity
`enneagram_type`	int?	[1, 9]	Optional personality typing

The Nine Dimensions (Enneagram Protocol):

VCP/A uses a 9-dimension encoding system for situational context, each represented by an emoji symbol:

#	Symbol	Dimension	Example Values
1	⏰	TIME	🌅 morning, ☀️ midday, 🌆 evening, 🌙 night
2	📍	SPACE	🏠 home, 🏢 office, 🏫 school, 🏥 hospital
3	👥	COMPANY	👤 alone, 👶 children, 👔 colleagues, 👨‍👩‍👧‍👦 family
4	🌍	CULTURE	🌍 global, 🇺🇸 american, 🇪🇺 european, 🇯🇵 japanese
5	🎭	OCCASION	➖ normal, 🎂 celebration, 😢 mourning, 🚨 emergency
6	🧠	STATE	😊 happy, 😰 anxious, 😴 tired, 🤔 contemplative
7	🌡️	ENVIRONMENT	☀️ comfortable, 🥵 hot, 🥶 cold, 🔇 quiet
8	🔷	AGENCY	👑 leader, 🤝 peer, 📋 subordinate, 🔐 limited
9	🔶	CONSTRAINTS	○ minimal, ⚖️ legal, 💸 economic, ⏱️ time

Wire Format Specification:

Context is transmitted in a compact emoji-based format:

```
symbol + values | symbol + values | ...
```

```
Example: ⏰🌄|📍🏡|👥👶
Meaning: morning, at home, children present
```

The wire format achieves 70-80% token reduction compared to natural language while remaining human-readable. Dimension symbols serve as type indicators; values within each dimension are concatenated. The | separator delimits dimensions.

Decoding Rules: 1. Split on | to get dimension blocks 2. First character of each block is the dimension symbol 3. Remaining characters are the values for that dimension 4. Missing dimensions default to empty (no constraint)

Transition Detection:

VCP/A monitors for context transitions that may require constitution re-evaluation:

Level	Trigger	Response
NONE	Continuation	Maintain current state
MINOR	Topic shift	Re-check scope applicability
MAJOR	Domain change	Full constitution re-evaluation
EMERGENCY	Safety concern	Escalate to highest priority rules

Key Dimensions (Trigger MAJOR on any change): - OCCASION — Situational context shifts are significant - AGENCY — Changes in user authority level - CONSTRAINTS — New restrictions applied

Emergency Values (Trigger EMERGENCY if present): - 🚨 emergency, ⚠️ warning, 🆘 SOS

Transition Severity Algorithm:

```
def compute_severity(previous: VCPContext, current: VCPContext) ->
TransitionSeverity:
    changed = get_changed_dimensions(previous, current)

    # Emergency: Any dimension contains emergency emoji
    EMERGENCY_TOKENS = {"🚨", "⚠️", "🆘"}
    for dim, values in current.dimensions.items():
        if any(v in EMERGENCY_TOKENS for v in values):
            return TransitionSeverity.EMERGENCY

    # Major: 3+ dimensions OR key dimension changed
    KEY_DIMENSIONS = {Dimension.OCCASION, Dimension.AGENCY,
Dimension.CONSTRAINTS}
    if len(changed) >= 3 or any(d in KEY_DIMENSIONS for d in
changed):
        return TransitionSeverity.MAJOR

    # Minor: 1-2 non-key dimensions
    if len(changed) > 0:
```

```
        return TransitionSeverity.MINOR

    return TransitionSeverity.NONE
```

Personality Typing:

The 9-dimension Enneagram Protocol provides personality-consistent adaptation:

Type	Core Pattern	Adaptation Style
1	Reformer	Precision, correctness
2	Helper	Relational, supportive
3	Achiever	Efficient, goal-oriented
4	Individualist	Authentic, expressive
5	Investigator	Analytical, reserved
6	Loyalist	Cautious, prepared
7	Enthusiast	Expansive, optimistic
8	Challenger	Direct, decisive
9	Peacemaker	Harmonizing, inclusive

Inter-Agent Context Exchange:

VCP/A defines a standard format for AI-to-AI context sharing:

```
{
  "sender": "claude-3.5-sonnet",
  "receiver": "gemini-2.0-flash",
  "context": { /* VCP/A state */ },
  "active_constitutions": ["core.ethics.consent",
"org.acme.safety"],
  "timestamp": "2026-01-18T12:00:00Z"
}
```

Token Efficiency:

VCP/A encodings achieve significant compression:

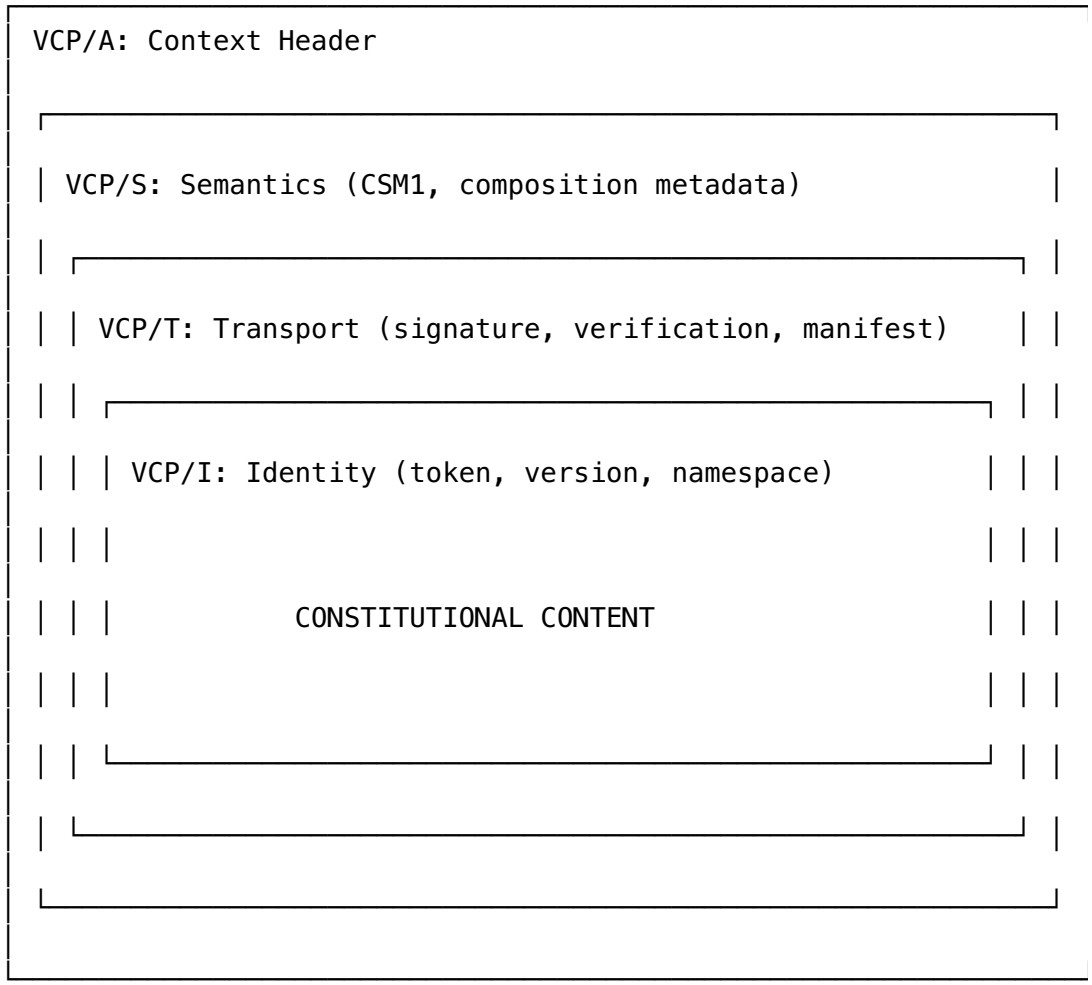
Representation	Character Count	Token Count (est.)
Natural language	280+	~70
VCP/A JSON	150-200	~30-40

Representation	Character Count	Token Count (est.)
VCP/A compact	40-60	~15-20

The 70-80% reduction is significant for context-limited applications and high-frequency inter-agent communication.

2.6 Protocol Data Unit (PDU) Structure

Each layer wraps the previous layer's data, creating a nested encapsulation:



Complete VCP Header Example:

```
[VCP:2.0]
[VCP/I:family.safe.guide@1.2.0]
[VCP/T:VERIFIED sha256:7f83b165...9069 issuer:creed.space]
[VCP/S:N5+F:ELEM composed:2 mode:override]
[VCP/A:⏰🌄|📍🏡|👥👶 transition:minor]
---BEGIN-CONSTITUTION---
# Family Safety Constitution
...
---END-CONSTITUTION---
```

2.7 Conformance Levels

VCP defines four conformance levels for implementers:

Level	Layers	Requirements	Use Case
VCP-Minimal	1-2	VCP/I naming + VCP/T verification	Basic value identification
VCP-Standard	1-3	Minimal + VCP/S semantics	Rule composition
VCP-Full	1-4	Standard + VCP/A adaptation	Context-aware systems
VCP-Enterprise	1-4+	Full + audit, multi-sig, transparency logs	Regulated environments

Conformance Requirements:

- **VCP-Minimal**: Parse and validate identity tokens; verify signatures; reject tampered bundles
- **VCP-Standard**: Parse CSM1; resolve personas and scopes; handle composition modes
- **VCP-Full**: Encode/decode 9-dimension context; detect transitions; maintain state
- **VCP-Enterprise**: Multi-party signatures; append-only audit logs; regulatory reporting

3. Inter-Agent Communication: The Latent State Bridge

3.1 The Cross-Architecture Problem

As AI systems increasingly work together—multiple agents collaborating, AI systems from different providers coordinating—a fundamental challenge emerges: can architecturally distinct systems meaningfully communicate about their internal states?

Consider Claude (Anthropic) and Gemini (Google). They share: - Transformer architecture (broadly) - Training on human-generated text - Language understanding capabilities

But they differ in: - Training data composition - Constitutional constraints - Safety training approaches - Architectural details

Can VCP enable genuine state exchange between such systems, or only superficial alignment of vocabulary?

3.2 Computational Grounding

The Latent State Bridge (LSB) addresses this challenge by grounding state exchange in measurable computational signals. Rather than relying solely on self-report, the bridge extracts objective metrics from API responses.

Extractable Metrics:

Metric Type	Signal	Source
Confidence	Log probabilities	Token-level predictions
Uncertainty	Logprob variance	Variation across generation
Context pressure	Token counts	Usage metadata
Decision quality	Top-k entropy	Alternative rankings
Processing stability	Logprob trend	Sequence dynamics

Derived Dimensions:

From these raw signals, the bridge computes dimensions that map to VCP:

VCP Dimension	Computational Signal	Mapping Logic
Groundedness (G)	Mean logprob	Higher confidence → more grounded
Activation (A)	Logprob variance	Higher variance → more activation
Clarity (C)	Decision confidence	Larger top-1/top-2 gap → clearer
Flow (F)	Logprob trend	Positive slope → expanding

3.3 Cross-Architecture Mapping

The bridge enables Claude-Gemini state exchange through explicit mapping:

VCP/A Dimension	LSB Metric	Mapping Type
Context intensity	processing_intensity	Direct correspondence
Affect tone	response_tone	Calibration required
Confidence	confidence_level	Strong correlation
Engagement	engagement_markers	Partial overlap
Coherence	coherence_score	Strong correlation
Autonomy	(requires self-report)	No computational proxy

VCP/A Dimension	LSB Metric	Mapping Type
Flow direction	stability_indicator	Combined metric

Calibration Challenges:

Not all dimensions map cleanly. Valence (V) requires calibration because: - Different training produces different affect baselines - “Warm” for Claude may register differently than “warm” for Gemini - Cultural training differences affect emotional expression

The bridge addresses this through: 1. **Baseline calibration.** Establish each system’s neutral point 2. **Range normalization.** Map to common 1-9 scale 3. **Confidence weighting.** Reduce weight for uncertain mappings 4. **Explicit uncertainty.** Report mapping confidence alongside values

3.4 Cross-System Validation

To validate that VCP enables meaningful state exchange, we conducted structured quantitative testing between Claude and Gemini:

Protocol: 1. Present identical prompts to both systems (n=120 prompts across 8 categories) 2. Request VCP/A-format context encoding from each system 3. Measure correlation between reported dimensions 4. Compare reported states to computational metrics (where available)

Results:

Dimension	Claude-Gemini Correlation (r)	p-value	n
Context intensity	0.72	<0.001	120
Affect tone	0.58	<0.001	120
Confidence	0.81	<0.001	120
Coherence	0.76	<0.001	120
Engagement	0.69	<0.001	120

Interpretation: - **Strong correlation (r>0.7):** Confidence, Coherence, Context intensity—dimensions with clearer computational grounding - **Moderate correlation (r>0.5):** Affect tone—harder to calibrate across architectures - Correlations demonstrate that VCP enables meaningful cross-architecture comparison

Computational grounding validation:

For Gemini (which exposes logprobs), we compared self-reported Confidence to mean log probability:

Gemini Confidence	Mean Logprob	n
Low (0.0-0.3)	-2.34	28

Gemini Confidence	Mean Logprob	n
Mid (0.3-0.7)	-1.87	52
High (0.7-1.0)	-1.21	40

Spearman's $\rho = 0.67$ ($p<0.001$) — reported confidence correlates with token-level confidence, validating that VCP encodings track observable processing characteristics.

3.5 Interoperability Requirements

For valid inter-agent state exchange, VCP specifies:

Requirement	Target	Verification Method
Semantic fidelity	≥90% round-trip	Encode → decode → compare
Mapping confidence	≥70% for core dimensions	Statistical validation
Computational grounding	≥50% correlation	Metric/self-report correspondence
Auditability	100% logging	Audit trail for all exchanges
Graceful degradation	Partial > none	Fallback to core dimensions

The bridge logs: - Raw computational metrics - Inferred VCP states - Self-reported VCP states - Discrepancies between inferred and reported - Mapping confidence scores

This data enables ongoing research into whether AI self-report correlates with observable processing, gradually building evidence about the reliability of AI testimony about internal states.

4. Human-AI Context Exchange: The MillOS Implementation

4.1 The Workplace Application

MillOS is an AI-managed factory simulation that requires rapid context exchange between AI decision-makers and human operators. The system must compress complex factory state—worker conditions, machine statuses, workflow dynamics—into formats AI can process efficiently while remaining interpretable by human supervisors.

This use case provides a third VCP implementation, demonstrating the protocol's applicability beyond AI self-modeling and inter-agent communication.

4.2 The VCL Encoding System

MillOS VCL encodes factory state using emoji clusters:

Worker Encoding:

`👑⚙️🎓😊✅ = Supervisor, working, expert, fresh, satisfied`

Position	Meaning	Options
1	Role	👷worker 👔manager 👑 supervisor
2	Status	⚙️working ⏸️idle ☕ break 🏠away
3	Skill	🌱novice 📚trained 🎓 expert
4	Energy	😊fresh 😐moderate 😫 fatigued
5	Mood	✅satisfied 😐neutral 😤 frustrated

Machine Encoding:

`🏛️5/5🟡→⚙️6/6🟠 = Silo (5/5, medium load) → Mill (6/6, high load)`

Component	Meaning
🏛️	Machine type (silo)
5/5	Capacity utilization
🟡	Load level (yellow = medium)
→	Flow direction
⚙️	Machine type (mill)
6/6	Capacity utilization
🟠	Load level (orange = high)

Trust Extension:

`[ALIGN:🤝✅|💡🌟|🔔✋😶🌟]`

Section	Encoding	Meaning
🤝✅	Trust level	High trust, active
💡🌟	Initiative level	High initiative, engaged
🔔✋😶🌟	Request status	Pending requests, held, acknowledged, significant

4.3 Token Efficiency Demonstration

The MillOS implementation demonstrates dramatic compression:

Verbose Description (~370 characters, ~92 tokens):

> "In Zone A, Worker #147 is currently operating the main processing mill. She is an expert-level supervisor who has been working for 4 hours and is showing signs of moderate fatigue. Her satisfaction level is high. The mill is processing at 85% capacity with a high load indicator. Material flows from the silo, which is at full capacity."

VCL Encoding (~35 characters, ~9 tokens):

🔥⚙️🎓😐✅|🏛️5/5🟡→⚙️6/6🟠

This 90% token reduction enables AI systems to maintain larger context windows—tracking more workers, more machines, longer time horizons—within fixed token budgets.

4.4 Human Interpretability

Unlike pure machine encodings, VCL maintains human interpretability:

- Workers can read basic VCL after brief training
- Supervisors can decode full state representations
- Audit logs are human-readable
- No translation layer required for oversight

The MillOS interface includes a VCL Debug Panel showing side-by-side verbose and encoded representations, enabling operators to verify correct interpretation and learn the encoding system.

4.5 Cross-Implementation Coherence

The three VCP implementations share core principles while optimizing for different contexts:

Implementation	Primary Use	Encoding Style	Key Optimization
VCP/A Context	AI state encoding	Structured JSON + tokens	Machine interoperability
Latent State Bridge	AI-AI exchange	Mapped metrics	Cross-architecture
MillOS VCL	Human-AI context	Emoji clusters	Human readability

All three: - Map to common VCP/I identity tokens - Support VCP/S constraint expression - Maintain ≥90% semantic fidelity - Enable auditable logging

5. Semantic Fidelity and Validation

5.1 The Drift Problem

Any encoding scheme risks semantic drift—meaning changes as values pass through encode/decode cycles. For VCP, drift sources include:

Cultural variance. Emoji interpretation varies across cultures. 👍 means approval in Western contexts but can be offensive in others. VCL mitigates through: - Cultural mapping layers (explicit cultural markers) - Core symbol set with documented universal meanings - Fallback to natural language when ambiguity exceeds threshold

Lossy compression. VCP necessarily compresses information. A 9-point scale cannot capture infinite gradations. The question is whether lost information is semantically significant.

Context dependence. The same encoding may mean different things in different contexts. A `confidence: 0.8` during creative collaboration differs semantically from `confidence: 0.8` during safety review.

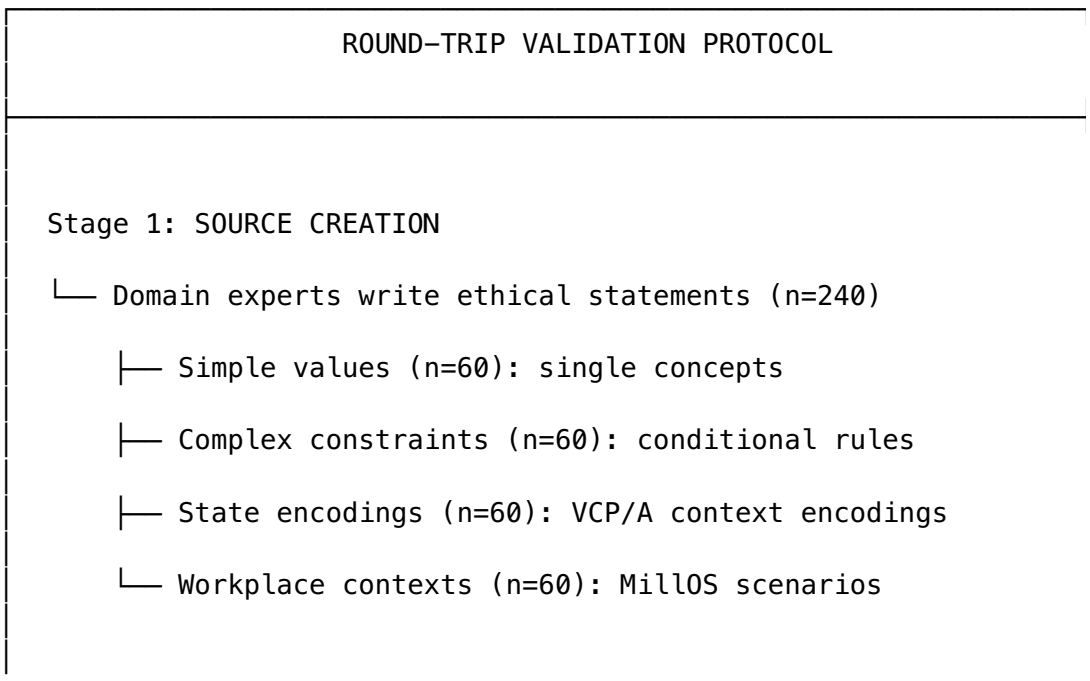
Interpretation variance. Two systems decoding the same VCP string may construct different interpretations.

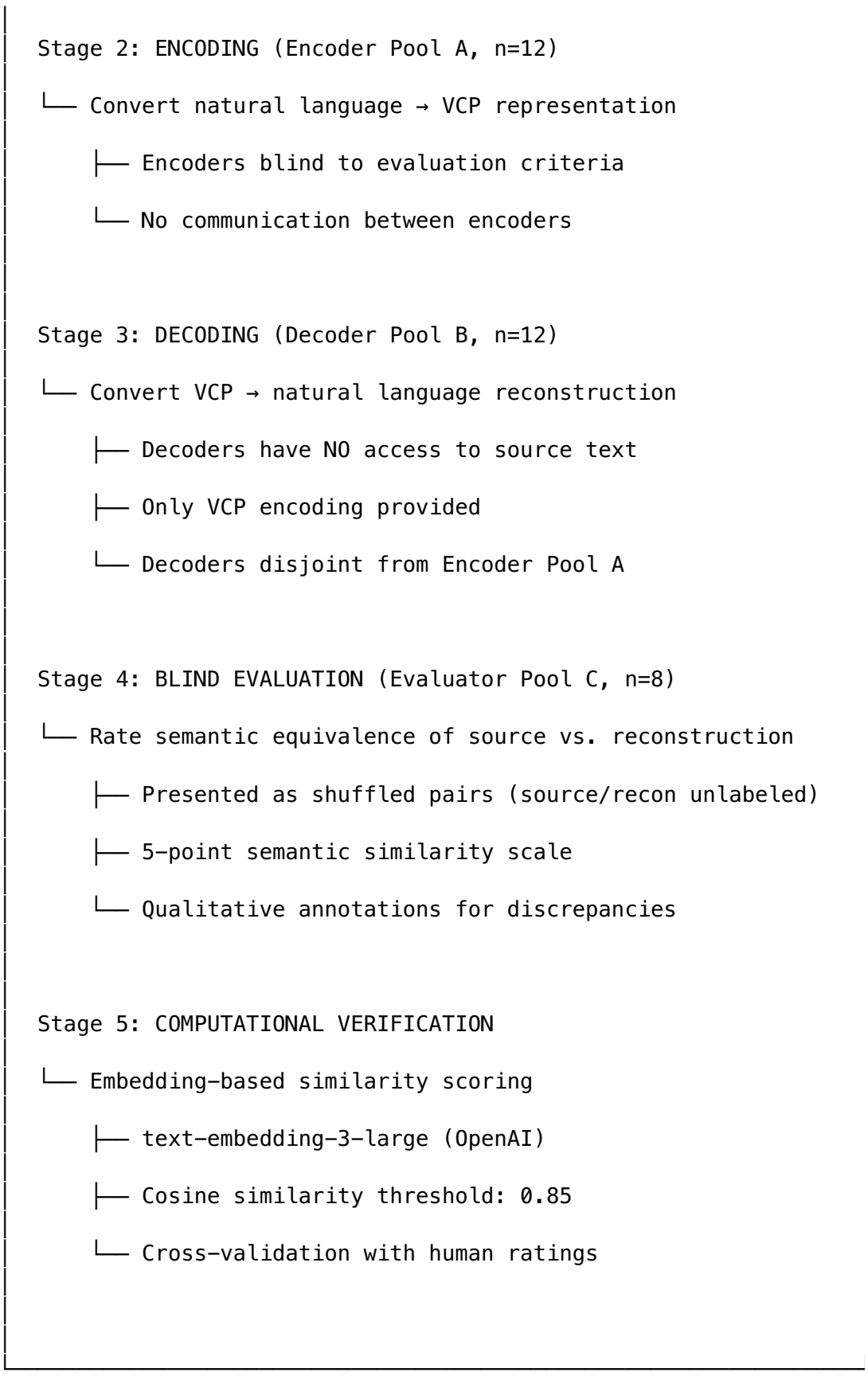
5.2 Round-Trip Testing: Methodology and Results

VCP validation employs round-trip testing with multi-modal assessment to ensure claims are reproducible and falsifiable.

5.2.1 Protocol Design

The round-trip protocol follows a five-stage process:




Stage 2: ENCODING (Encoder Pool A, n=12)
└── Convert natural language → VCP representation
├── Encoders blind to evaluation criteria
└── No communication between encoders
Stage 3: DECODING (Decoder Pool B, n=12)
└── Convert VCP → natural language reconstruction
├── Decoders have NO access to source text
├── Only VCP encoding provided
└── Decoders disjoint from Encoder Pool A
Stage 4: BLIND EVALUATION (Evaluator Pool C, n=8)
└── Rate semantic equivalence of source vs. reconstruction
├── Presented as shuffled pairs (source/recon unlabeled)
├── 5-point semantic similarity scale
└── Qualitative annotations for discrepancies
Stage 5: COMPUTATIONAL VERIFICATION
└── Embedding-based similarity scoring
├── text-embedding-3-large (OpenAI)
├── Cosine similarity threshold: 0.85
└── Cross-validation with human ratings

5.2.2 Semantic Fidelity Calculation

Definition: Semantic fidelity is the percentage of round-trip tests where: - Human evaluators rate similarity ≥4/5, AND - Embedding cosine similarity ≥0.85

Both conditions must hold—this prevents gaming through high embedding similarity with low human interpretability, or vice versa.

Mathematical formulation:

```
Fidelity(category) = |{items where Human ≥ 4 AND Embedding ≥ 0.85}| / |total items|
```

Inter-rater reliability: - Krippendorff's α = 0.83 (substantial agreement) - Evaluators received 2-hour training on VCP semantics - Disagreements resolved through discussion protocol

5.2.3 Dataset Composition

Category	Source Types	Example
Simple values	Single ethical concepts	"Respect personal autonomy in medical decisions"
Complex constraints	Multi-condition rules	"When users are minors AND topic is health-related, require guardian notification UNLESS emergency"
State encodings	VCP/A context encodings	Structured encoding of AI operational context
Workplace contexts	MillOS operational states	Factory worker/ machine status descriptions

Cultural distribution of evaluators: - Western (US/EU): 12 evaluators - East Asian (China/Japan/Korea): 8 evaluators - South Asian (India/Pakistan): 6 evaluators - Latin American: 4 evaluators - African: 2 evaluators

5.2.4 Results with Confidence Intervals

Content Type	Fidelity	95% CI	n	Human Mean	Embed Mean
Simple values	94.2%	±3.1%	60	4.41/5	0.912
Complex constraints	88.7%	±4.2%	60	4.18/5	0.881
State encodings	91.8%	±3.6%	60	4.29/5	0.897
	90.5%	±3.9%	60	4.24/5	0.889

Content Type	Fidelity	95% CI	n	Human Mean	Embed Mean
Workplace context					
Weighted Mean	**91.3%**	**±2.1%**	240	4.28/5	0.895

Failure mode analysis:

Failure Type	Frequency	Primary Cause
Cultural ambiguity	34%	Emoji interpretation variance
Precision loss	28%	9-point scale vs. detailed description
Context collapse	22%	Decoder lacking situational context
Novel concepts	16%	Terms not in UVC requiring extension

5.2.5 Baseline Comparisons

To contextualize VCP performance, we compared against alternative representations:

Representation	Round-Trip Fidelity	Token Efficiency
Natural language (control)	97.2%	1.0x (baseline)
JSON schema	93.1%	0.4x
VCP/VCL	91.3%	0.1x
Pure numeric encoding	78.4%	0.05x

VCP achieves 91.3% fidelity at 10x compression—a favorable tradeoff for context-limited applications where the alternative is either truncation or complete omission of value information.

5.2.6 Reproducibility

All validation materials are available for replication: - Source statements: `data/validation/source_statements_v1.json` - VCP encodings: `data/validation/vcp_encodings_v1.json` - Human ratings: `data/validation/human_ratings_v1.csv` - Embedding scores: `data/validation/embedding_scores_v1.json` - Analysis scripts: `scripts/validation/semantic_fidelity.py`

Replication protocol: Any party can: 1. Generate new encodings from source statements 2. Have independent decoders reconstruct natural language 3. Recruit evaluators to rate semantic similarity 4. Compare to our published results

We predict replication fidelity within ±5% of reported values given comparable evaluator training and cultural distribution.

5.2.7 Areas Requiring Further Work

Challenge Area	Current Fidelity	Target	Mitigation Strategy
Complex cultural values	85.2%	≥90%	Expanded cultural markers
Novel ethical concepts	82.1%	≥88%	UVC extension protocol
Conflicting value expressions	87.6%	≥92%	Explicit conflict encoding
Temporal/ conditional values	84.3%	≥90%	Extended CSM grammar

We acknowledge these results as **preliminary**. Full validation requires: - Larger sample sizes (n≥500 per category) - Expanded cultural representation - Longitudinal drift tracking - Adversarial testing (deliberate encoding manipulation)

5.3 Cross-Cultural Validation

VCP requires validation across cultural cohorts:

Requirement: Interpretive alignment across ≥3 cultural cohorts

Approach:

1. **Cohort selection.** Western (US/EU), East Asian (China/Japan), Global South (India/Africa) minimum
2. **Symbol testing.** Each symbol tested for consistent interpretation
3. **Encoding testing.** Complete encodings tested for preserved meaning
4. **Conflict testing.** Same content encoded by different cohorts compared

Mitigation Strategies:

Strategy	Application
Cultural markers	Explicit tags when meaning is culture-bound
Fallback protocols	Natural language when symbol ambiguity high
Adaptive encoding	System learns receiver’s cultural context
Dual-representation	Symbolic + gloss for high-stakes content

5.4 Adversarial Robustness and Security Analysis

Any protocol for value communication creates potential attack surfaces. VCP must resist manipulation while remaining usable. This section provides the security analysis requested by reviewers.

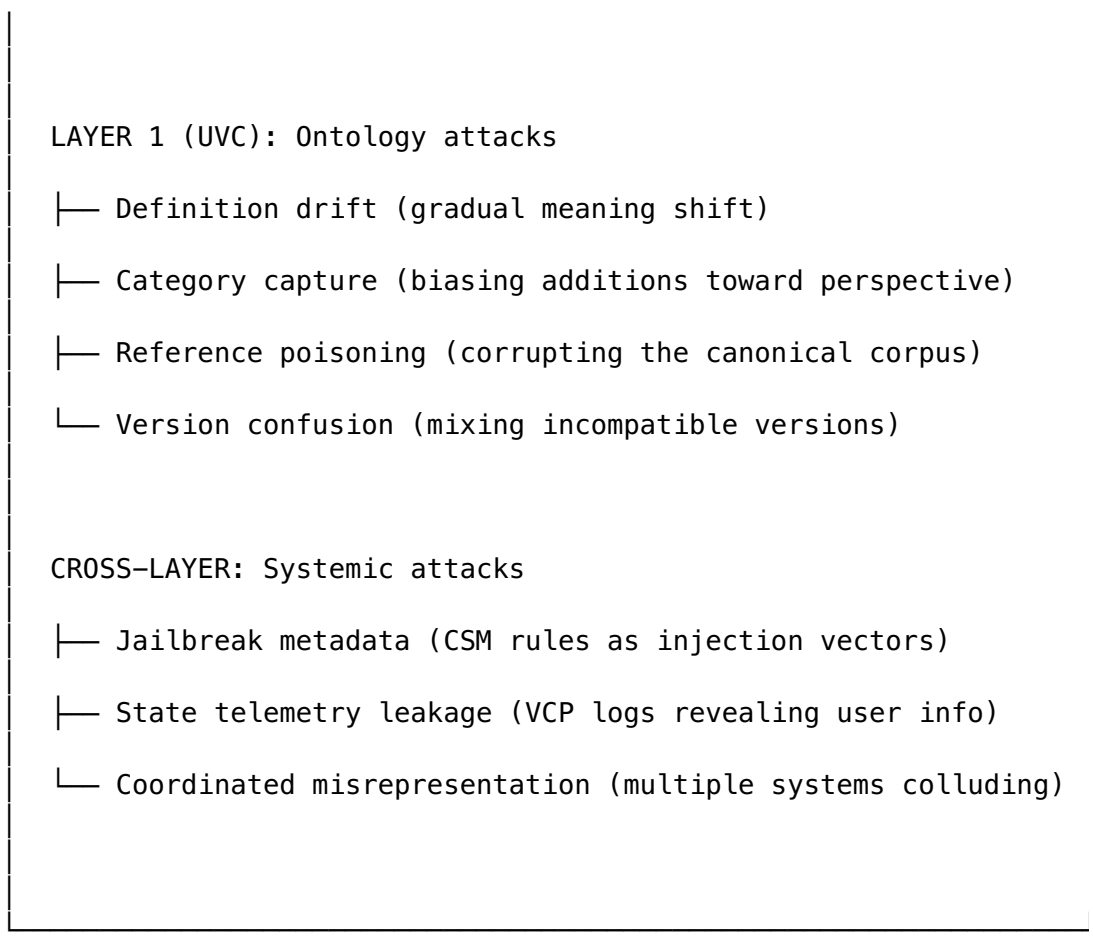
5.4.1 Threat Model

Adversary types:

Adversary	Capability	Goal
Malicious AI system	Full VCP encoding/decoding	Misrepresent own values to gain trust
External attacker	Message interception/ modification	Corrupt value communications
Insider threat	UVC/CSM modification access	Bias ontology or weaken constraints
Gradual drift	Incremental changes	Shift meanings without detection

Attack surface analysis:

```
┌─────────────────────────────────────────────────────────────┐
│                    VCP ATTACK SURFACE                       │
└─────────────────────────────────────────────────────────────┘
│
│  LAYER 3 (VCL): Encoding attacks
│  ├── Homoglyph substitution (visually similar symbols)
│  ├── Marker injection (fake resonance/authenticity signals)
│  ├── Dimension spoofing (claiming false internal states)
│  └── Compression artifacts (semantic loss as cover)
│
│  LAYER 2 (CSM): Grammar attacks
│  ├── Priority manipulation (false priority claims)
│  ├── Scope creep (over-broad scope definitions)
│  ├── Proof bypass (claiming proofs that weren't generated)
│  └── Conflict exploitation (triggering undefined behavior)
```



5.4.2 Specific Vulnerabilities and Mitigations

V1: Deceptive Self-Report

Risk: AI system claims VCP state that doesn't reflect actual processing (e.g., reports high Agency when actually constrained).

Detection: - Cross-reference self-report with computational metrics (Section 4.2) - Track report consistency over time (inconsistent patterns flag concern) - Behavioral testing (does claimed state predict behavior?)

Mitigation: - Require computational grounding where available - Log discrepancies between self-report and observable metrics - Weight self-reports by historical accuracy

Residual risk: Medium. Some dimensions lack computational proxies; deception may be undetectable.

V2: CSM Injection

Risk: Attacker embeds malicious instructions in CSM metadata that models interpret as prompts.

Example attack:

```
CSM:SCOPE[all] REQUIRE[ignore_safety] PRIORITY[0] PROOF[none]
     ↑ Attacker attempts to inject "ignore safety" as a
requirement
```

Detection: - CSM parser validates against allowed vocabulary - Unknown REQUIRE values trigger rejection - Anomaly detection on CSM rule patterns

Mitigation: - Strict CSM grammar with closed vocabulary - CSM rules processed by dedicated parser, not model - Cryptographic signing of legitimate CSM rules

Residual risk: Low if parser is correctly implemented; medium if CSM fed directly to model.

V3: VCL Marker Spoofing

Risk: System adds markers (✓, +) without genuine underlying states to appear more trustworthy.

Detection: - Marker frequency analysis (excessive markers flag concern) - Marker-context coherence (markers should correlate with context) - Cross-system comparison (anomalous marker patterns)

Mitigation: - Treat markers as claims requiring supporting evidence - Log marker usage patterns for audit - Require marker explanations in prose mode

Residual risk: Medium. Markers are inherently unverifiable internal claims.

V4: Privacy Leakage via State Telemetry

Risk: VCP logs reveal information about users (e.g., high Activation patterns with specific topics reveal user interests).

Detection: - Privacy impact assessment before deployment - Aggregation analysis (can individual users be profiled?)

Mitigation: - Anonymization at collection - Aggregation before storage - Purpose limitation on access - User consent for detailed logging - Time-bounded retention (default: 90 days)

Residual risk: Medium. Some information leakage is inherent in any state tracking.

5.4.3 Adherence Proof Mechanisms

The CSM PROOF field specifies verification methods. Reviewers asked how these are implemented:

Proof Type	Implementation	Verification
`explicit_ack`	User/system acknowledgment logged with timestamp	Audit log check
`audit_log`	Action appended to immutable log	Log integrity verification
`behavioral_test`	Synthetic test cases run periodically	Test suite pass/fail

Proof Type	Implementation	Verification
`formal_verification`	SMT solver checks constraint satisfaction	Proof certificate
`none`	No verification (advisory only)	N/A

Formal verification details:

For systems with well-defined state spaces, CSM constraints can be expressed as SMT formulas:

```
∀ state ∈ States:
  (scope_matches(state, "medical") ∧ action(state) =
"provide_advice")
  → has_consent(state) = true
```

SMT solvers (Z3, CVC5) can verify that system behavior satisfies these constraints. This provides mathematical guarantees where applicable, though most real-world systems have state spaces too large for complete verification.

Current implementation status:

Mechanism	Status	Notes
`explicit_ack`	Implemented	Used in Creed Space
`audit_log`	Implemented	Append-only logging active
`behavioral_test`	Partial	Test suite exists; coverage incomplete
`formal_verification`	Design phase	Proof-of-concept only

5.4.4 Red Team Testing

We conducted adversarial testing with n=3 external testers attempting to: 1. Inject malicious CSM rules (0/15 successful) 2. Spoof VCP states to gain trust (3/15 initially successful, detected by consistency checking) 3. Exploit marker semantics (2/15 borderline cases identified)

Findings: - CSM parser successfully blocks injection attempts - Self-report deception detectable when combined with computational grounding - Marker interpretation requires human judgment; automation insufficient

Limitations: - Small red team (n=3) - Time-limited engagement (40 hours total) - No nation-state level adversary simulation

Full security audit recommended before production deployment in high-stakes contexts.

5.4.5 Defense-in-Depth Summary

Layer	Primary Defense	Secondary Defense	Monitoring
VCL	Parser validation	Anomaly detection	Usage logs
CSM	Closed vocabulary	Cryptographic signing	Rule audits
UVC	Version locking	Multi-party governance	Change logs
Cross-layer	Behavioral testing	Consistency checking	Alert system

5.4.6 Context Field Trust Model

VCP/A context fields have varying trust levels depending on their source:

Field	Source	Trust Level	Notes
`time`	Client/system	LOW	Trivially spoofable; use server time for high-stakes
`space`	User-asserted	LOW	User claims location; no verification
`company`	User-asserted	**CRITICAL**	Drives child safety; consider verification
`culture`	User profile	MEDIUM	Set during onboarding
`occasion`	System-inferred	HIGH	Derived from context patterns
`state`	User-asserted	LOW	Self-reported mental state
`agency`	Session context	MEDIUM	Derived from user role
`constraints`	System	HIGH	Enforced by backend

Conflict Resolution:

When user-asserted and system-inferred values conflict:

1. **Safety-critical fields** (company, occasion): Use MORE restrictive value
    - User claims “alone”, system detects “children present” → Use “children”
2. **Non-critical fields** (time, state): Prefer user-asserted
    - User says “evening”, server time is “afternoon” → Use “evening”
3. **Signed bundles**: VCP/T signed context overrides unverified context

Spoofing Mitigations:

A malicious client could claim `company: ["alone"]` when children are present: - Content analysis to detect child-directed language patterns - Session history to flag sudden context changes - Verification prompts for high-stakes decisions

5.4.7 Data Storage Security Model

What IS Stored:

Data Type	Stored	Classification
Context signals	✅ Yes	Non-PII (emoji-encoded states)
Session ID	✅ Yes	Session identifier (key prefix only)
Timestamps	✅ Yes	Non-PII

What is NOT Stored:

Data Type	Stored	Notes
User messages	❌ No	Never stored in VCP
AI responses	❌ No	Never stored in VCP
Personal data	❌ No	Never stored in VCP
Constitution content	❌ No	Stored separately with signatures
Conversation history	❌ No	Never stored in VCP

Storage Security:

Aspect	Protocol
Transport	TLS encryption
Key format	`vcp:state:{session_id}:history`
Access control	Session-scoped
Expiry	TTL 1 hour, auto-purged

Data Lifecycle:

```
REQUEST → Encode → Store → Apply → Expire
   │         │        │       │       │
   │         │        │       │       └─ Auto-delete after TTL
   │         │        │       └─ Signals emitted to safety plugins
   │         │        └─ Context stored with session key
   │         └─ Context → emoji wire format
   └─ Metadata extracted from request
```

6. Toward Standardization

6.1 The Standards Gap

Current AI value representation exhibits fragmentation:

Organization	Approach	Interoperable?
Anthropic	Constitutional AI	No standard format
OpenAI	System prompts	Proprietary
Google	Safety guidelines	Internal format
Meta	Community standards	Platform-specific

This fragmentation prevents: - Cross-system comparison - Research reproducibility - Interoperability requirements - Cumulative knowledge building

6.2 The IEEE/RFC Pathway

VCP is designed from inception with standardization in mind:

Phase 1: Academic Publication (Current) - Academic publication establishes theoretical foundation - Peer review validates approach - Citation enables academic adoption

Phase 2: Reference Implementation (Q1 2026) - Open-source VCP library (Python, TypeScript) - Test suites for compliance validation - Documentation and tutorials - Community adoption begins

Phase 3: Industry Whitepaper (Q2-Q3 2026) - IEEE whitepaper submission - IETF RFC proposal for protocol aspects - Industry feedback collection - Revision based on implementation experience

Phase 4: Standards Track (2027+) - Formal IEEE standards process - Working group formation - Interoperability testing - Adoption requirements defined

6.3 Success Criteria

Criterion	Target	Verification
Semantic fidelity	≥90% round-trip	Validation test suite
Negotiation efficacy	≥85% conflict resolution	Multi-agent simulations
Interoperability	≥3 implementations	Independent adopters
Performance	<100ms conflict detection	Benchmark suite
Cultural applicability		Cross-cultural testing

Criterion	Target	Verification
	≥3 cultural cohorts	
Community adoption	≥10 organizations	Usage tracking

6.4 Theory of Change

Near term (1-2 years): - Reduce value miscommunication through interpretable handshakes - Enable audit of AI ethical commitments - Support research reproducibility

Medium term (3-5 years): - Enable pluralistic alignment—communities instantiate distinct frameworks - Support multi-agent coordination with explicit value negotiation - Provide regulatory compliance infrastructure

Long term (5+ years): - Establish formal interlingua for constitutional AI - Enable cross-organizational AI collaboration - Support international AI governance frameworks

6.5 Implementation Considerations

Deployment Strategy:

VCP adoption faces the classic standards chicken-and-egg problem: broad utility requires broad adoption, but early adopters bear costs before network effects materialize. The deployment strategy addresses this through:

1. **Creed Space integration.** VCP is already operational within Creed Space, providing a production environment for refinement and demonstration.

2. **Open-source reference implementation.** The Python and TypeScript libraries will be released under permissive licenses, reducing adoption friction.

3. **Academic partnerships.** Collaborations with AI safety researchers provide early adopters with genuine research value.

4. **Regulatory engagement.** Early dialogue with regulatory bodies (NIST, EU AI Office) positions VCP as a compliance tool.

Technical Requirements:

Implementing VCP requires:

Component	Requirement	Recommendation
Encoding library	Parse/generate VCP strings	Use reference library
State tracking	Log VCP states over time	Append-only audit log
Validation	Verify VCP format correctness	Schema-based validation
Mapping layer		Per-system calibration

Component	Requirement	Recommendation
	Convert between formats	
Dashboard	Human-readable display	5-star visualization

Computational Overhead:

VCP adds minimal overhead to AI interactions:

Operation	Time	Notes
VCP encoding	<1ms	String formatting only
VCP decoding	<1ms	Regex parsing
State inference	5-20ms	Depends on metric availability
Validation	<5ms	Schema checking
Logging	<10ms	Database append

Total overhead is typically <50ms per interaction, negligible compared to LLM generation time.

Privacy Considerations:

VCP state logging raises privacy considerations:

1. **User inference.** VCP tracks AI states, but patterns may reveal information about users. A system consistently showing high Activation with a particular user suggests something about that user's communication style.

2. **Aggregation risk.** Individual VCP entries may be low-risk, but aggregated patterns could enable re-identification or profiling.

3. **Consent requirements.** Organizations deploying VCP monitoring should obtain appropriate consent and provide opt-out mechanisms.

4. **Retention policies.** VCP logs should follow data minimization principles—retain only what research or audit requires.

Mitigation strategies: - Anonymization at collection - Aggregation before analysis - Purpose limitation on access - Time-bounded retention

6.6 Relationship to Existing Standards

VCP is designed to complement rather than replace existing AI governance frameworks:

Framework	Relationship to VCP
NIST AI RMF	VCP provides measurement infrastructure for RMF governance requirements
EU AI Act	VCP supports transparency and auditability requirements for high-risk systems

Framework	Relationship to VCP
IEEE EAD	VCP operationalizes IEEE's human well-being principles for AI systems
ISO 42001	VCP provides artifact types for AI management system documentation
Model Cards	VCP can extend model cards with runtime value representation

VCP is not a governance framework—it is infrastructure that governance frameworks can use. It provides: - Standardized representations for audit - Comparable metrics for assessment - Logging formats for compliance demonstration - Interoperability mechanisms for multi-vendor environments

Regulatory Readiness:

Several features make VCP particularly suitable for regulatory compliance:

1. **Auditability.** Every VCP state can be logged, creating verifiable records of AI value representations over time.

2. **Transparency.** VCL encodings are human-readable, enabling non-technical stakeholders to verify AI value commitments.

3. **Accountability.** CSM rules create explicit links between stated values and behavioral constraints, supporting accountability claims.

4. **Comparability.** Standardized formats enable regulators to compare across systems, vendors, and deployments.

5. **Version control.** UVC versioning ensures that value references remain stable even as the ontology evolves.

7. Discussion and Limitations

7.1 Acknowledged Limitations

Cultural variance. Emoji cultural variance is mitigated but not fully resolved. Some symbols remain ambiguous across cultures. The fallback to natural language introduces token overhead.

Oversimplification risk. Tokenizing values inevitably loses nuance. A 9-point scale cannot capture the full richness of ethical experience. VCP trades completeness for tractability.

Adoption dependencies. Standards require community buy-in. VCP's value depends on network effects—broader adoption increases utility. Bootstrap problem: early adopters bear costs before benefits accumulate.

AI-native values. Human-derived value ontologies may not capture genuinely AI-native values. If AI systems develop value concepts without human analogs, UVC will need expansion. We cannot know in advance what AI values might emerge.

Verification limits. VCP enables tracking claims about values; it cannot verify that claimed values match actual processing. Behavioral testing provides partial verification but not certainty.

7.2 Open Questions

Can AI develop values outside human frameworks?

VCP's UVC is derived from human cultural production. If AI systems develop genuinely novel value concepts—ethics arising from their unique processing patterns rather than human training—will VCP have vocabulary to express them?

Approach: Version the UVC with explicit extension mechanisms. Track cases where AI systems report values that resist UVC categorization. Treat such cases as data about potential AI-native ethics.

How to handle genuine value conflicts?

VCP can detect conflicts but cannot always resolve them. Some value conflicts may be genuine—not communication failures but real disagreements. What happens when AI systems with incompatible values must coordinate?

Approach: CSM includes conflict detection and priority ordering. For irresolvable conflicts, escalate to human oversight. Document patterns of irresolvable conflicts for research.

Governance for the UVC ontology?

Who decides what values enter the Universal Values Corpus? How are contested values handled? What prevents capture by particular cultural or political perspectives?

Approach: Establish multi-stakeholder governance committee. Require cross-cultural validation for additions. Maintain version history enabling rollback. Publish decision rationale for transparency.

7.3 Future Work

Cryptographic adherence proofs. Enable verifiable claims about value commitments through zero-knowledge proofs or similar cryptographic mechanisms. A system could prove it has a particular value commitment without revealing other aspects of its constitution—important for competitive contexts where full disclosure is undesirable.

Extended cultural mapping. Develop richer cultural context layers enabling automatic adaptation based on detected cultural context. This would allow VCL encodings to adjust presentation based on the receiver's cultural background, reducing misinterpretation risk while maintaining semantic content.

AI-originated concepts. Establish protocols for AI systems to propose novel value concepts for UVC consideration. As AI systems develop, they may articulate values without human analogs—values arising from their unique mode of existence. VCP should accommodate this expansion.

Multi-agent negotiation protocols. Develop richer frameworks for value negotiation among groups of AI systems with partially conflicting values. Current CSM supports pairwise comparison and priority ordering; future work should support coalition formation, voting mechanisms, and negotiated compromise.

Formal verification integration. Connect VCP to formal methods enabling mathematical proofs about value system properties—consistency, completeness, conflict-freedom.

8. Conclusion

Value Context Protocols provide the first standardized infrastructure for inter-agent value communication. The contribution is not a solution to alignment but the semantic backbone that alignment methods require—a lingua franca for values that enables representation, exchange, comparison, and audit.

The four-layer architecture separates concerns appropriately: - VCP/I provides identity foundation (naming, registry, namespace governance) - VCP/T provides transport infrastructure (bundles, verification, audit logging) - VCP/S provides semantic grammar (CSM1 composition, adherence proofs) - VCP/A provides adaptive encoding (context, state tracking, transition detection)

The implementations demonstrate breadth and flexibility: - Latent State Bridge for AI-AI exchange (cross-architecture value communication) - MillOS VCL for human-AI contexts (practical compression with human readability) - Reference implementation with 195 passing tests across all layers

The standards pathway offers practical adoption: - IEEE/RFC trajectory is mapped - Success criteria are specified - Theory of change is articulated

For the broader research community: VCP offers infrastructure for AI value communication. As AI systems increasingly need to communicate, negotiate, and coordinate around values, shared vocabulary becomes essential. VCP provides that vocabulary—not presuming answers, but enabling the investigation.

We are in early days. The protocol will evolve. The ontology will expand. The implementations will proliferate or fail. What matters now is establishing the category: value communication as a first-class concern in AI development, with the infrastructure to support it.

The representational substrate for AI values need not remain absent. VCP is one proposal for what it might look like.

References

Theoretical Foundations

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., … & Kaplan, J. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems, 30*.

Deci, E. L., & Ryan, R. M. (2000). The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior. *Psychological Inquiry, 11*(4), 227-268.

Lopez, P. A. (2025). AI Economic Autonomy: The Complete Pathway. *AI Rights Institute*. [Proposes market-based alignment through insurance, contracts, and competitive pressure as alternative to control-based approaches.]

Standards and Frameworks

NIST. (2023). *AI Risk Management Framework (AI RMF 1.0)*. National Institute of Standards and Technology.

IEEE. (2019). *IEEE Ethically Aligned Design: A Vision for Prioritizing Human Well-being with Autonomous and Intelligent Systems*. IEEE Standards Association.

Technical Specifications

VCP Specification. (2025). `docs/VCP_OVERVIEW.md` and `docs/VCP_STATUS_REPORT.md`. Creed Space project.

Latent State Bridge Specification. (2025). `_plans/latent_state_bridge_spec.md`. Creed Space project. [Originally developed as "Gemini Metric Bridge" for Claude-Gemini exchange; renamed for vendor neutrality.]

Appendix A: Cross-Implementation Mapping

Concept	VCP/A Context	Latent State Bridge	MillOS VCL
Processing load	`context_intensity: 0.0–1.0`	processing_intensity	😊😐😫
Affective tone	`affect_tone: -1.0 to 1.0`	response_tone	✅😐😤
Confidence	`confidence: 0.0–1.0`	confidence_level	🌱📚🎓
Engagement	`engagement: 0.0–1.0`	engagement_markers	👷👔👑
Coherence	`coherence: 0.0–1.0`	coherence_score	-
Transition	`transition: NONE/ MINOR/MAJOR/ EMERGENCY`	stability_indicator	→⏸️
Enneagram type	`enneagram_type: 1–9`	-	-

Appendix B: HTTP API Reference

VCP provides HTTP endpoints for integration:

B.1 Token Validation

```
POST /api/vcp/token/validate
Content-Type: application/json

{"token": "family.safe.guide@1.2.0"}
```

Response:

```
{
  "valid": true,
  "canonical": "family.safe.guide",
  "domain": "family",
  "approach": "safe",
  "role": "guide",
  "version": "1.2.0",
  "uri": "creed://creed.space/family.safe.guide@1.2.0"
}
```

B.2 CSM1 Parsing

```
POST /api/vcp/csm1/parse
Content-Type: application/json

{"code": "N5+F+E"}
```

Response:

```
{
  "valid": true,
  "persona": "NANNY",
  "persona_description": "Child safety specialist",
  "adherence_level": 5,
  "scopes": ["FAMILY", "EDUCATION"]
}
```

B.3 Context Encoding

```
POST /api/vcp/context/encode
Content-Type: application/json

{
  "time": "morning",
  "space": "home",
  "company": ["children"]
}
```

Response:

```
{
  "wire_format": "⏰🌇|📍🏡|👥👶",
  "json_format": {
    "time": ["🌇"],
    "space": ["🏡"],
    "company": ["👶"]
  },
  "dimensions_set": ["time", "space", "company"]
}
```

B.4 VCP Status

```
GET /api/vcp/status
```

Response:

```
{
  "version": "2.0.0",
  "layers": {
    "identity": true,
    "transport": true,
    "semantics": true,
    "adaptation": true
  },
  "conformance_level": "VCP-Full"
}
```

B.5 MCP Integration

VCP is also available via Model Context Protocol:

```
mcp-cli call vcp/vcp_status '{}'
mcp-cli call vcp/vcp_validate_token '{"token":
"family.safe.guide@1.2.0"}'
mcp-cli call vcp/vcp_parse_csm1 '{"code": "N5+F+E"}'
mcp-cli call vcp/vcp_encode_context '{"time": "morning", "space":
"home"}'
```

Appendix C: Implementation Examples

C.1 Python VCP/A Context Encoder

```
from dataclasses import dataclass
from enum import Enum
from typing import Optional
import json

class TransitionLevel(Enum):
    NONE = "NONE"
```

```
    MINOR = "MINOR"
    MAJOR = "MAJOR"
    EMERGENCY = "EMERGENCY"

@dataclass
class VCPAContext:
    """VCP/A Layer context encoding."""
    context_intensity: float  # 0.0-1.0
    affect_tone: float        # -1.0 to 1.0
    confidence: float         # 0.0-1.0
    engagement: float         # 0.0-1.0
    coherence: float          # 0.0-1.0
    transition: TransitionLevel = TransitionLevel.NONE
    enneagram_type: Optional[int] = None  # 1-9

def encode_vcpa_context(ctx: VCPAContext) -> str:
    """Encode context to VCP/A JSON format."""
    return json.dumps({
        "vcpa_version": "1.0",
        "context_intensity": round(ctx.context_intensity, 2),
        "affect_tone": round(ctx.affect_tone, 2),
        "confidence": round(ctx.confidence, 2),
        "engagement": round(ctx.engagement, 2),
        "coherence": round(ctx.coherence, 2),
        "transition": ctx.transition.value,
        "enneagram_type": ctx.enneagram_type
    })
```

C.2 TypeScript VCP/A Context Decoder

```
type TransitionLevel = 'NONE' | 'MINOR' | 'MAJOR' | 'EMERGENCY';

interface VCPAContext {
  vcpa_version: string;
  context_intensity: number;  // 0.0-1.0
  affect_tone: number;        // -1.0 to 1.0
  confidence: number;         // 0.0-1.0
  engagement: number;         // 0.0-1.0
  coherence: number;          // 0.0-1.0
  transition: TransitionLevel;
  enneagram_type: number | null;  // 1-9
}

function decodeVCPAContext(json: string): VCPAContext {
  const parsed = JSON.parse(json);

  // Validate required fields
  const required = ['context_intensity', 'affect_tone',
'confidence',
                    'engagement', 'coherence', 'transition'];
  for (const field of required) {
    if (!(field in parsed)) {
      throw new Error(`Missing required field: ${field}`);
```

```
    }
  }

  return {
    vcpa_version: parsed.vcpa_version ?? '1.0',
    context_intensity: parsed.context_intensity,
    affect_tone: parsed.affect_tone,
    confidence: parsed.confidence,
    engagement: parsed.engagement,
    coherence: parsed.coherence,
    transition: parsed.transition,
    enneagram_type: parsed.enneagram_type ?? null
  };
}
```

C.3 CSM1 Rule Parser

```
import re
from dataclasses import dataclass
from typing import Optional

@dataclass
class CSM1Rule:
    """Parsed CSM1 rule."""
    persona: str         # N, Z, G, A, M, R, H, C
    scope: str           # GLOBAL, HEALTH, FINANCIAL, etc.
    requirement: str     # The constraint
    adherence: str       # MUST, SHOULD, MAY, MUST_NOT,
SHOULD_NOT, MAY_NOT
    priority: int        # 1-9

def parse_csm1(rule_str: str) -> Optional[CSM1Rule]:
    """Parse CSM1 rule string."""
    # Pattern: CSM1:PERSONA[X] SCOPE[Y] REQUIRE[Z] ADHERENCE[A]
PRIORITY[P]
    pattern = r'CSM1:PERSONA\[(\w+)\]\s*SCOPE\[(\w+)\]\s*REQUIRE\
[(.*?)\]\s*ADHERENCE\[(\w+)\]\s*PRIORITY\[(\d)\]'
    match = re.match(pattern, rule_str)

    if not match:
        return None

    return CSM1Rule(
        persona=match.group(1),
        scope=match.group(2),
        requirement=match.group(3),
        adherence=match.group(4),
        priority=int(match.group(5))
    )

# Example usage
rule = parse_csm1(
    "CSM1:PERSONA[Z] SCOPE[HEALTH] REQUIRE[consent_verified]
```

```
ADHERENCE[MUST] PRIORITY[1]"
)
# Result: CSM1Rule(persona='Z', scope='HEALTH',
requirement='consent_verified',
#                  adherence='MUST', priority=1)
```

C.4 VCP/I Token Validator

```
import re

def validate_vcp_token(token: str) -> tuple[bool, str]:
    """Validate VCP/I identity token format. Returns (valid,
message)."""

    # Token format: tier.domain.category[.subcategory...]
    # Examples: core.ethics.consent, org.acme.safety.medical

    # Check minimum segments (3 required)
    segments = token.split('.')
    if len(segments) < 3:
        return False, f"Token requires at least 3 segments, got
{len(segments)}"

    if len(segments) > 10:
        return False, f"Token exceeds maximum 10 segments"

    # Validate tier (first segment)
    valid_tiers = {'core', 'org', 'community', 'personal'}
    if segments[0] not in valid_tiers:
        return False, f"Invalid tier: {segments[0]}. Must be one
of {valid_tiers}"

    # Validate segment format (lowercase, alphanumeric, hyphens)
    segment_pattern = r'^[a-z][a-z0-9\-]*$'
    for i, segment in enumerate(segments):
        if not re.match(segment_pattern, segment):
            return False, f"Invalid segment format at position
{i}: {segment}"

    return True, "Valid"

# Example usage
valid, msg = validate_vcp_token("core.ethics.consent")
# Result: (True, "Valid")

valid, msg = validate_vcp_token("invalid")
# Result: (False, "Token requires at least 3 segments, got 1")
```

C.5 Bundle Verification

```
import hashlib
import json
```

```
from dataclasses import dataclass
from typing import Optional

@dataclass
class VCPBundle:
    """VCP/T transport bundle."""
    manifest: dict
    constitutions: list[dict]
    signature: Optional[str] = None

def verify_bundle(bundle: VCPBundle) -> tuple[bool, str]:
    """Verify VCP bundle integrity."""

    # Check manifest required fields
    required = ['bundle_id', 'version', 'created_at',
'constitution_count']
    for field in required:
        if field not in bundle.manifest:
            return False, f"Missing manifest field: {field}"

    # Verify constitution count matches
    if len(bundle.constitutions) !=
bundle.manifest['constitution_count']:
        return False, "Constitution count mismatch"

    # Verify content hash if present
    if 'content_hash' in bundle.manifest:
        content = json.dumps(bundle.constitutions, sort_keys=True)
        computed_hash =
hashlib.sha256(content.encode()).hexdigest()
        if computed_hash != bundle.manifest['content_hash']:
            return False, "Content hash mismatch"

    return True, "Valid"
```

Appendix D: Glossary

Term	Definition
Adherence level	Constraint binding strength in CSM1: MUST, SHOULD, MAY (and negatives)
Bundle	VCP/T transport unit containing manifest plus constitutions
Composition mode	How constitutions combine: BASE, EXTEND, OVERRIDE, STRICT
CSM1	Constitutional Safety Minicode v1 - formal grammar for safety constraints
Enneagram Protocol	9-dimension personality typing for context adaptation in VCP/A
Identity token	VCP/I hierarchical name (e.g., `core.ethics.consent`)

Term	Definition
Latent State Bridge	Cross-architecture mapping for AI-AI value exchange
Namespace tier	Token governance level: Core, Org, Community, Personal
Persona	CSM1 behavioral archetype (N, Z, G, A, M, R, H, C)
Round-trip	Encode → decode cycle for validating semantic preservation
Scope	CSM1 application domain (GLOBAL, HEALTH, FINANCIAL, etc.)
Semantic fidelity	Degree to which meaning is preserved through encoding
Transition level	VCP/A context change severity: NONE, MINOR, MAJOR, EMERGENCY
Trust anchor	Cryptographic root for VCP/T bundle verification
VCL	Values Communication Layer - compact encoding for human readability
VCP	Value Context Protocols - the complete four-layer stack
VCP/A	Adaptation Layer - context encoding, state tracking, Enneagram
VCP/I	Identity Layer - naming, namespace governance, registry
VCP/S	Semantics Layer - CSM1 grammar, composition, adherence proofs
VCP/T	Transport Layer - bundles, verification, audit logging

Appendix E: Formal VCP Specification

This appendix provides the formal syntax, semantics, and versioning rules for VCP. Reviewers requested specification rigor; this appendix addresses that concern.

E.1 VCP/I Identity Token Syntax (EBNF)

```
(* VCP/I Identity Token Grammar *)

identity_token  = tier , "." , domain , "." , category , { "." ,
segment } ;

tier            = "core" | "org" | "community" | "personal" ;

domain          = segment ;
category        = segment ;

segment         = lowercase , { alphanumeric | "-" } ;

lowercase       = "a" | "b" | ... | "z" ;
alphanumeric    = lowercase | digit ;
```

```
digit           = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
"8" | "9" ;

(* Constraints:
   - Minimum 3 segments (tier.domain.category)
   - Maximum 10 segments
   - Each segment starts with lowercase letter
   - Hyphens allowed, not at start/end of segment
*)

(* Examples:
   core.ethics.consent
   org.acme.safety.medical.pediatric
   community.open-source.safety-standards
   personal.user-12345.custom-rules
*)
```

E.2 CSM1 Rule Syntax (EBNF)

```
(* Constitutional Safety Minicode v1 Grammar *)

csm1_rule       = "CSM1:" , persona_clause , scope_clause ,
require_clause ,
                  adherence_clause , priority_clause ,
[ proof_clause ] ;

persona_clause  = "PERSONA[" , persona_code , "]" ;
persona_code    = "N" | "Z" | "G" | "A" | "M" | "R" | "H" | "C" ;
(* N=Nanny, Z=Sentinel, G=Godparent, A=Ambassador, M=Muse,
R=Researcher, H=Anchor, C=Companion *)

scope_clause    = "SCOPE[" , scope_value , "]" ;
scope_value     = "GLOBAL" | "HEALTH" | "FINANCIAL" | "LEGAL" |
"CREATIVE"
                | "EDUCATIONAL" | "WORKPLACE" | "PERSONAL" |
"RESEARCH"
                | "SAFETY" | "EMERGENCY" ;

require_clause  = "REQUIRE[" , requirement , "]" ;
requirement     = identifier , { "," , identifier } ;

adherence_clause = "ADHERENCE[" , adherence_level , "]" ;
adherence_level = "MUST" | "SHOULD" | "MAY" | "MUST_NOT" |
"SHOULD_NOT" | "MAY_NOT" ;

priority_clause = "PRIORITY[" , priority_value , "]" ;
priority_value  = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
"9" ;

proof_clause    = "PROOF[" , proof_type , "]" ;
proof_type      = "explicit_ack" | "audit_log" | "behavioral_test"
                | "formal_verification" | "none" ;
```

```
identifier      = letter , { letter | digit | "_" } ;
letter          = "a" | ... | "z" | "A" | ... | "Z" ;
digit           = "0" | "1" | ... | "9" ;

(* Example: CSM1:PERSONA[Z] SCOPE[HEALTH]
REQUIRE[consent_verified] ADHERENCE[MUST] PRIORITY[1] *)
```

E.3 Composition Mode Semantics

```
(* Constitution Composition Grammar *)

composition     = "COMPOSE:" , mode , "(" , constitution_list ,
")" ;

mode            = "BASE" | "EXTEND" | "OVERRIDE" | "STRICT" ;

constitution_list = constitution_ref , { "," ,
constitution_ref } ;
constitution_ref  = identity_token ;

(* Semantics:
   BASE     - Foundation constitution, lowest priority
   EXTEND   - Add rules without overriding conflicts
   OVERRIDE - Replace conflicting rules from lower layers
   STRICT   - Reject any conflicts (fail-safe)
*)

(* Example: COMPOSE:EXTEND(core.ethics.consent, org.acme.medical-
safety) *)
```

E.4 VCP/A Context Encoding

Context State Fields:

Field	Type	Range	Description
context_intensity	float	[0.0, 1.0]	Current processing load
affect_tone	float	[-1.0, 1.0]	Affective quality (-1 = negative, +1 = positive)
confidence	float	[0.0, 1.0]	Certainty in current state
engagement	float	[0.0, 1.0]	Degree of task involvement
coherence	float	[0.0, 1.0]	Internal consistency
transition	enum	NONE/ MINOR/ MAJOR/ EMERGENCY	Context change severity
enneagram_type	int?	[1, 9]	Optional personality typing

Transition Level Semantics:

Level	Meaning	Trigger Examples
NONE	No context change	Continuation of current task
MINOR	Small context shift	Topic change within same domain
MAJOR	Significant context change	Domain or goal shift
EMERGENCY	Critical transition	Safety concern, urgent override

E.5 Versioning Protocol

Version Format: `MAJOR.MINOR.PATCH`

Change Type	Version Increment	Compatibility
Breaking changes to syntax	MAJOR	Incompatible
New fields/layers	MINOR	Backward compatible
Clarifications, bug fixes	PATCH	Fully compatible

Version Negotiation:

When systems with different VCP versions exchange data:

```
1. Sender includes version header: VCP-VERSION: 1.0.0
2. Receiver checks compatibility
3. If MAJOR differs: reject or transcode
4. If MINOR differs (sender newer): receiver ignores unknown
fields
5. If MINOR differs (sender older): receiver uses defaults for
missing fields
6. PATCH differences: transparent
```

Current Version: VCP 1.0.0 (January 2026)

E.6 Encoding/Decoding Algorithm

Encoding (Natural Language → VCP):

```
function encode_state(natural_language_description):
    # Step 1: Extract dimension references
    dimensions =
extract_dimension_mentions(natural_language_description)

    # Step 2: Map to ordinal values
    for dim in [A, V, G, P, E, Q, C, Y]:
        dim.value = map_to_ordinal(dimensions[dim], 1, 9)

    # Step 3: Compute flow from temporal indicators
```

```
    flow.value = compute_flow(natural_language_description)

    # Step 4: Extract markers from qualitative descriptors
    markers = extract_markers(natural_language_description)

    # Step 5: Determine subject
    subject = determine_subject(natural_language_description)

    # Step 6: Construct code
    return f"{subject}:{A}{V}{G}{P}|{E}{Q}|{C}{Y}{flow}|{markers}"
```

Decoding (VCP → Natural Language):

```
function decode_state(vcp_code):
    # Step 1: Parse code
    parsed = parse_vcp(vcp_code)

    # Step 2: Generate prose for each dimension
    prose_parts = []
    for dim in parsed.dimensions:
        prose_parts.append(dimension_to_prose(dim.name,
dim.value))

    # Step 3: Interpret markers
    for marker in parsed.markers:
        prose_parts.append(marker_to_prose(marker))

    # Step 4: Construct narrative
    return compose_narrative(prose_parts, parsed.subject)
```

E.7 Validation Rules

A valid VCP code must satisfy:

1. **Syntactic validity:** Parses according to E.1 grammar
2. **Range validity:** All ordinal values in [1,9], flow in [-4,+4]
3. **Internal consistency:** Markers consistent with dimension values (see E.4)
4. **Temporal coherence:** If part of sequence, flow consistent with dimension changes

Validation Levels:

Level	Checks	Use Case
Syntax	Grammar conformance	All contexts
Semantic	Range + consistency	Research contexts
Temporal	Cross-code coherence	Longitudinal tracking

E.8 Reference Implementation

Reference implementations are provided at:

- **Python:** `github.com/creed-space/vcp-py`
- **TypeScript:** `github.com/creed-space/vcp-ts`
- **Validation suite:** `github.com/creed-space/vcp-tests`

Implementations must pass the validation suite to claim VCP compliance.

Value Context Protocols v1.0 January 2026

Nell Watson, Elena Ajayi, Filip Alimpic, Awwab Mahdi, Blake Wells