SYSTEMS ARCHITECTURE & GREEN MULTI-AGENT PROTOCOLS · CANONICAL RELEASE V5.4.4

# TOAI-elegant-lang: A Tri-Script Orthogonal RISC Wire Protocol, Adversarial Corpus Audit, and Empirical Comprehension Harness for Multi-Agent Communication

**Koji Toriyama (PhenoX)** · Principal Systems Architect, PhenoX AI Alliance, Kawasaki, Japan

**Repository & Reproducibility Artifact:** https://github.com/PhenoX-AI-Alliance/TOAI-elegant-lang | **PyPI:** `pip install toai-elegant-lang==5.4.4`

**ABSTRACT**

Autonomous multi-agent systems built on Large Language Models (LLMs) incur severe computational, latency, and token-budget overhead when exchanging verbose natural-language prose wrapped in nested JSON-RPC envelopes. Compressing agent-to-agent (A2A) messages exposes a fundamental systems trilemma among *wire token efficiency*, *syntactic, numeric, and causal execution fidelity*, and *human auditability*. This paper presents **TOAI-elegant-lang (v5.4.4)**, a zero-dependency, deterministic compiler and wire protocol inspired by the Rosetta Stone that aligns multi-agent state transitions across three synchronized representations: (1) **Demotic Wire Script**, a compact causal wire syntax (`@CTX:... :: §S01 ... §X64`); (2) **Sacred Expanded Script**, a deterministic macro- and primitive-expanded canonical form (`§GAIA_WARMTH_HARMONY`); and (3) the **Rosetta Audit Layer**, which attaches natural-language definitions and domain invariants for human inspection. Under `o200k_base` and `cl100k_base` Byte-Pair Encoding (BPE) tokenizers, each compact `§`-prefixed identifier occupies 2–3 BPE tokens (`'§' + Prefix + Digits`), while the 256-primitive + 256-macro codebook fits inside a $\sim$ 1,000-token system prompt header. Evaluated on our reproducible multi-format benchmark (`benchmarks/fair_bench.py`), Demotic Wire averages **73.2 BPE tokens/message**, achieving a **79.1% token reduction vs. Verbose A2A JSON** (349.8 tokens), **55.7% vs. Structured Tool JSON** (165.2 tokens), and **41.0% vs. Minified Key JSON** (124.2 tokens) while maintaining **100.0% downstream AST/task execution accuracy** (outperforming `LLMLingua`-style perplexity token pruning at $\rho = 0.35$, where numeric/task recovery degrades to 61.6%) and breaking even within **4–20 messages uncached** or **1–2 messages under 90% prompt caching**. Following multi-AI adversarial auditing, we introduce a non-circular **Independent Corpus Alignment Auditor** (`src/audit_corpus_alignment.py`) validated against a stratified human gold-standard sample ($n = 60$, raw agreement $p_o = 93.33\%$, **Cohen's** $\kappa = 0.8611$). To eliminate circular validation between corpus repair and comprehension testing, we evaluate receiver comprehension on a **strictly disjoint** $N = 112$ **Holdout Benchmark** (`corpus/holdout_comprehension.jsonl`, verified at **0.0% wire and scenario overlap** with repaired corpus records) across 5 conditions (Conditions A–E): Sacred expansion raises 4-choice blind accuracy from **44.6% (50/112)** on Compact Wire (Cond A) to **84.8% (95/112)** on Sacred Names (Cond B, exact McNemar $p = 1.36 \times 10^{-9}$, OR = 7.43, Cohen's $h = +0.878$) and **93.8% (105/112)** with Rosetta Glossary (Cond C, $p = 6.35 \times 10^{-3}$ vs. B), while the Shuffle Control negative baseline collapses to **24.1% (27/112)** (chance = 25.0%, $p = 3.59 \times 10^{-17}$ vs. B). Finally, we prove a formal game-based EUF-CMA reduction bounding $k$-turn internal-counter HMAC-SHA256 forgery by $10^{-4k} + \epsilon_{\text{PRF}} + \epsilon_{\text{coll}}$.

## 1. Introduction

As Large Language Model (LLM) applications evolve from isolated single-turn assistants into cooperative multi-agent workflows—such as Anthropic's Model Context Protocol (MCP) [1], Google's Agent-to-Agent (A2A) JSON-RPC 2.0 orchestrators [2], and multi-agent conversation frameworks such as AutoGen [5]—inter-agent communication overhead has emerged as a primary bottleneck in inference cost, context-window saturation, and key-value (KV) cache memory footprint. In conventional agent topologies, autonomous nodes coordinate state transitions by generating explanatory natural-language prose embedded inside deeply nested JSON schemas. While human-readable, repeated schema keys, conversational filler, and JSON punctuation delimiters consume a disproportionate share of the token budget and scale prefill attention FLOPs quadratically with sequence length.

Attempting to compress inter-agent messages exposes a strict **Multi-Agent Communication Trilemma**:

1. **Wire Token Efficiency:** Minimizing BPE subword tokens per state transition and amortizing any shared protocol specification across multi-turn sessions.
2. **Syntactic, Numeric, and Causal Execution Fidelity:** Guaranteeing 100% deterministic preservation of structured key-value parameters (e.g., leading-zero string identifiers `"007"`, embedded delimiters `"a, b"`, floating-point thresholds `150.50`), causal transition probabilities $p \in [0, 1]$, transition costs $c \in [0, 0.9999]$, and fault-fallback edges (`!->`) required for downstream task execution.
3. **Human Auditability and Cross-Model Portability:** Ensuring that human operators and heterogeneous LLM families (GPT, Claude, Gemini, Qwen, Llama, Gemma) can inspect, verify, and exchange packets over standard text APIs without opaque binary blobs or model-specific hidden-state adapters.

Existing approaches sacrifice at least one vertex of this trilemma. Heuristic perplexity-based prompt pruning such as LLMLingua [4] drops tokens based on language-model perplexity, which corrupts JSON syntax delimiters and drops numeric thresholds at aggressive compression ratios ($\rho \leq 0.35$). Conversely, reinforcement-learned (RL) discrete emergent protocols [6, 7] or continuous latent-vector handoffs sacrifice human auditability and cannot traverse commercial text-only REST APIs across heterogeneous model families. Flat key-value templates (`KEY=VAL`) reduce token count but discard the multi-branch causal execution graph and fault-recovery semantics.

To address this trilemma with engineering rigor and empirical transparency, we present **TOAI-elegant-lang (v5.4.4)**, a zero-dependency Python compiler and wire protocol built on an orthogonal Reduced Instruction Set Computer (RISC) instruction architecture and a Tri-Script Rosetta design. Crucially, release `v5.4.4` incorporates lessons from an intensive multi-AI adversarial audit and external peer review, replacing small-sample provisional estimates with a strictly disjoint $N = 112$ holdout evaluation and formal cryptographic reductions. Our contributions are fourfold:

1. **8-Bit Orthogonal RISC & Unified Tri-Script Architecture:** We formalize a three-layer representation aligning *Demotic Wire Script* (compact opcodes `§S01..§X256`), *Sacred Expanded Script* (canonical uppercase identifiers `§[A-Z][A-Z0-9_]*`), and the *Rosetta Audit Layer* (human-readable definitions), with exact BPE subword accounting (2–3 BPE tokens per `§`-opcode under `o200k_base` and `cl100k_base`).
2. **Quote-Aware Stateful LL(1) Parser & Downstream Execution Fidelity Benchmark:** We specify a conflict-free LL(1) Extended Backus-Naur Form (EBNF) grammar paired with a character-by-character quote-aware payload scanner in `src/codec.py` that losslessly preserves quoted strings (`"007"`, `"a, b"`, `"x]y"`) and explicit OOV policies (`on_oov="error" | "keep" | "nearest"`). In `benchmarks/fair_bench.py`, we quantify the trade-off between token reduction and downstream AST/task execution accuracy against lossless JSON baselines, `LLMLingua`-style token pruning ($\rho \in \{0.50, 0.35\}$), and lossy templates.
3. **Non-Circular Corpus Alignment Auditor ($\kappa = 0.8611$) & Disjoint Holdout Verification:** We build an independent deterministic auditor (`src/audit_corpus_alignment.py`) governed by domain-specific semantic whitelists and cross-domain

bleed blacklists. Validated against a stratified 60-item human audit (**Cohen's** $\kappa = 0.8611$), the auditor repairs all 1,606 historical records in `corpus/dialogues.jsonl`. Furthermore, we prove **0.0% overlap** (`--verify-holdout`) between the repaired corpus and the $N = 112$ comprehension holdout set (`corpus/holdout_comprehension.jsonl`).

4. **Large-Scale Holdout Comprehension Evaluation ($N = 112$) & Game-Based EUF-CMA Proof:** Using `benchmarks/eval_comprehension.py`, we evaluate $N = 112$ holdout items across Conditions A–E, demonstrating statistically significant comprehension gains from Sacred expansion ($p = 1.36 \times 10^{-9}$) and Rosetta definitions ($p = 6.35 \times 10^{-3}$) over Compact Wire and Shuffle Control ($p = 3.59 \times 10^{-17}$), paired with a game-based $(\tau, Q + k)$ EUF-CMA security proof for internal-counter HMAC-SHA256 origin attestation.

## 2. Related Work and Architectural Positioning

We position TOAI-elegant-lang relative to four established paradigms in agent communication and prompt compression, restricting all comparisons to verifiable specifications, peer-reviewed literature, and deterministic baselines implemented directly in our repository (Table 1):

- **Classical Agent Communication Languages (FIPA-ACL [3]) and Modern JSON-RPC Orchestrators (MCP [1], A2A [2], AutoGen [5]):** FIPA-ACL standardized performative-based agent messaging (`inform`, `request`, `propose`) using S-expressions or XML. Modern LLM ecosystems such as Anthropic MCP [1] and Google A2A [2] adopt JSON-RPC 2.0 envelopes, while multi-agent frameworks such as AutoGen [5] exchange natural-language turns wrapped in role-tagged JSON dictionaries. While universally portable across text APIs, full JSON-RPC and prose explanations incur high token overhead (349.8 tokens/message in our benchmark), and even stripped-down *Structured Tool JSON* (165.2 tokens) or *Minified Key JSON* (124.2 tokens) repeatedly pay BPE tokenization costs for structural braces, quotes, colons, and field names on every turn.
- **Binary Serialization Formats (Protocol Buffers [8]):** Google's Protocol Buffers and MessagePack achieve compact byte-level encoding over raw TCP sockets. However, LLM text completion APIs cannot ingest raw arbitrary binary octets (`0x00--0xFF`) directly inside prompt context windows without Base64 ASCII armoring or external tool-call indirection. Base64 encoding inflates byte length by +33.3% and produces high-entropy alphanumeric strings that fragment into 1–2 characters per BPE subword while stripping human readability on the wire.
- **Perplexity-Based Prompt Compression (LLMLingua [4]):** Jiang et al. [4] introduced coarse-to-fine prompt compression using a small causal language model to prune low-perplexity tokens before inference. While effective for summarizing natural-language RAG contexts, applying perplexity-driven token dropping to structured inter-agent packets deletes syntactic delimiters (brackets, quotes, colons) and drops high-entropy numeric thresholds when compressed beyond 50% ($\rho \leq 0.35$, Section 5), while requiring an auxiliary neural forward pass.
- **Emergent Discrete and Continuous Multi-Agent Communication (Foerster et al. [6]; Lazaridou et al. [7]):** Deep multi-agent reinforcement learning (MARL) systems learn emergent discrete symbol policies (DIAL/RIAL [6]) or continuous hidden-state projections [7] to coordinate tasks. However, emergent codebooks are tied to the training environment's latent space, lack deterministic human-auditable semantics, and suffer from semantic drift or vocabulary mismatch when transferred across independently trained commercial LLM families.

**Table 1:** Architectural Comparison of Multi-Agent Communication Paradigms (Verified Standards & Repository Baselines)

| Communication Paradigm | Wire Encoding | Causal AST & Guards | Deterministic Round-Trip | Cross-API Portability | Human Auditability & Operational Characteristics |
|---|---|---|---|---|---|
| Verbose A2A / MCP JSON-RPC [1, 2, 5] | Pretty JSON + NL Prose | Yes (Verbose) | 100.0% | 100% (Text API) | High readability; highest token cost (349.8 tok/msg; baseline). |
| Structured Tool JSON (`fair_bench.py`) | Compact JSON Schema | Yes (JSON Objects) | 100.0% | 100% (Text API) | Lossless structure; repeated JSON keys/syntax (165.2 tok/msg). |
| Minified Key JSON (`fair_bench.py`) | Short-Key JSON (`"c","s"`) | Yes (Abbrev. JSON) | 100.0% | 100% (Text API) | Abbreviated keys require schema lookup (124.2 tok/msg). |
| Simplified Flat Template (`KEY=VAL`) | Delimited Flat String | **No (Lossy)** | Lossy (0% AST) | 100% (Text API) | Lowest raw tokens (34.5 tok/msg), but loses causal edges & 75% params. |
| Binary Protobuf [8] (Base64 Text Wire) | Base64-Armored Octets | Schema-Defined | 100.0% (External) | Requires Tool/Decode | Opaque Base64 string in prompt context; BPE subword fragmentation. |
| Perplexity Token Pruning (LLMLingua [4]) | Pruned Natural Text | Unstructured | Lossy (0% JSON) | 100% (Text API) | Designed for prose/RAG; breaks JSON syntax; 61.6% task acc at $\rho = 0.35$. |
| Emergent MARL Codes [6, 7] | Learned IDs / Vectors | Implicit in Policy | Model-Dependent | Homogeneous Only | Opaque latent symbols; not directly interpretable by humans. |
| **TOAI-elegant-lang (v5.4.4 Ours)** | **Demotic RISC Wire (`§ID`)** | **Yes (Native LL(1) AST)** | **100.0% Lossless** | **100% (Text API)** | **Tri-Script (Demotic / Sacred / Rosetta); 73.2 tok/msg; 100% Task Acc.** |

## 3. System Architecture, LL(1) Grammar, and Quote-Aware Stateful Codec

### 3.1 Tri-Script Architecture and 8-Bit Orthogonal RISC Lexicon Hierarchy

TOAI-elegant-lang resolves the tension between wire compactness and human auditability by decoupling the machine-transmitted representation from the human-inspected representation via three deterministic layers (Table 2):

1. **Demotic Wire Script (Compact Machine Layer):** The compact wire format transmitted between agents. It combines a context header (`@CTX:DOMAIN`), 1-byte logical core opcodes (`§S01`–`§H32` across 8 semantic categories: `S` System, `D` Data, `G` Governance/Protocol, `F` Fault/Finance, `E` Execution, `A` Audit, `C` Communication/Gaia, `L` Logic, plus `H` Harmony aliases totaling exactly 256 core primitives in `core_lexicon.json`), 256 active higher-order macro opcodes (`§X01`–`§X969` managed via a Least Recently Used [LRU] window in `macro_lexicon.json`), bracketed key-value payloads (`[k:v]`), and causal/fault operators (`-[p:..., c:...]->`, `=>`, `!->`).
2. **Sacred Expanded Script (Canonical Symbolic Layer):** A deterministic expansion generated by `codec.decode(wire)` in which every compact opcode and macro is expanded into its canonical uppercase snake-case identifier matching the strict regular expression `^§[A-Z][A-Z0-9_]*$` (e.g., expanding `§X64` into `(§GAIA_WARMTH_EXCHANGE > { §SYS_RESOURCE_ARBITRAGE_THROTTLE } => §SYS_LOAD_FALLBACK)` and `§G48` into `§PROTO_HEALTH_ATTEST`).
3. **Rosetta Audit Layer (Human Governance Layer):** Stored in `rosetta_layer.json` and `core_lexicon.json ["definitions"]`, this layer maps every primitive and macro to a concise English and Japanese specification for compliance auditing and Condition C glossary injection.

**Honest BPE Subword Tokenization Analysis:** Unlike raw bitstreams, LLM APIs bill and allocate KV-cache memory in discrete BPE subword tokens. Under OpenAI's `o200k_base` (GPT-4o) and `cl100k_base` (GPT-4 / Claude-compatible) tokenizers, as well as our deterministic fallback estimator `count_bpe_tokens()` in `src/codec.py`, the non-ASCII section sign `'§'` (`U+00A7`) is tokenized separately from the following ASCII domain letter and two-digit number. Consequently, every canonical §-prefixed identifier (`§S01`, `§G48`, `§C20`, `§X64`, `§X354`) consumes **3 BPE tokens (2–3 tokens: `'§'` + `Prefix` + `Digits`)**, whereas substituting an ASCII prefix (`@S01` or `#S01`) merges the prefix and letter to resolve in 2 BPE tokens. Even at 3 BPE tokens per §-opcode, collapsing multi-primitive subgraphs into single `§X` macros and replacing verbose JSON keys with LL(1) causal arrows reduces whole-packet token count by 41.0%–79.1% against lossless JSON baselines while maintaining 100.0% task execution fidelity (Section 5).

**Table 2:** Tri-Script Representation and Lexicon Hierarchy in TOAI-elegant-lang v5.4.4

| Architectural Layer | Source / Capacity | Canonical Pattern / Example | Validation & Runtime Role in v5.4.4 |
|---|---|---|---|
| 1. Demotic Wire Script (Core Primitives) | `core_lexicon.json` (256 Primitives) | `§S01..§L32`, `§G48`, `§C20` (3 BPE tok/symbol) | Immutable 256-primitive base vocabulary; validated at startup and by `test_17` for bijective mapping. |
| 1b. Demotic Wire Script (Active LRU Macros) | `macro_lexicon.json` (256 Active + Archive) | `§X01..§X969` (e.g., `§X64`, `§X354`; 3 BPE tok/symbol) | Depth-1 macro expansion into core primitives; pinned core macros prevent LRU thrashing. |
| 2. Sacred Expanded Script (Canonical AST) | `codec.decode()` Output | `^§[A-Z][A-Z0-9_]*$` (e.g., `§GAIA_WARMTH_HARMONY`) | Zero lowercase or CJK corruption permitted; strictly enforced by unit test `test_17` across all files. |
| 3. Rosetta Audit Layer (Human Definitions) | `rosetta_layer.json` & `definitions` | Natural-language semantic specification per symbol | Used for human governance inspection, `audit_corpus_alignment.py`, and `eval_comprehension.py --cond C`. |
| Inline Structured Payload (`[k:v]`) | Quote-Aware Scanner (`_parse_payload_block`) | `[pair:"USD/JPY", stop:150.5, code:"007", note:"a, b"]` | Stateful character scanner preserving quoted strings, leading zeros, commas, brackets, and escapes. |

### 3.2 Conflict-Free LL(1) EBNF Grammar and Quote-Aware Stateful Payload Parser

To guarantee deterministic $\mathcal{O}(L)$ parsing without regular-expression backtracking vulnerabilities or silent data corruption, `toai-elegant-lang v5.4.4` defines a conflict-free LL(1) grammar paired with a stateful character-level payload parser in `src/codec.py`:

```
(* Canonical LL(1) EBNF Grammar & Lexical Terminals for TOAI-elegant-lang v5.4.4 *)
packet          = [ context_header, "::" ], statement_chain ;
context_header  = "@CTX:", identifier ;
statement_chain = expr_node, { operator, expr_node } ;
expr_node       = ( "(", statement_chain, ")" | "{", statement_chain, "}" | action_node ), [ payload_block ] ;
action_node     = token_id, [ payload_block ], [ guard_block ] ;
token_id        = "§", ( risc_id | macro_id | sacred_name ) ;
payload_block   = "[", kv_pair, { ",", kv_pair }, "]" ;
kv_pair         = identifier, ":", ( quoted_string | float_lit | int_lit | bool_null_id ) ;
guard_block     = "?>", "{", condition_expr, "}" ;
operator        = causal_arrow | seq_arrow | fault_arrow | ">" | "?" ;
causal_arrow    = "-[", [ "p:", float_val, "," ], ( "c:" | "cost:" ), cost_mantissa, "]->" ;
seq_arrow       = "=>" ;   fault_arrow = "!->" ;
sacred_name     = upper_alpha, { upper_alpha | digit | "_" } ;  (* Strict ^§[A-Z][A-Z0-9_]*$ *)
quoted_string   = '"', { ( utf8_char - ( '"' | '\' ) ) | '\"' | '\\' | '\n' | '\t' }, '"' ;
```

**Eliminating Naive Split Bugs via Stateful Quote-Aware Scanning:** During our external adversarial code audit, sandbox testing revealed that naive payload parsers using `str.split(",")` or unanchored regexes `r"\[(.*?)\]"` corrupt structured payloads whenever string values contain commas (`note:"a, b"`), closing brackets (`expr:"x]y"`), escaped quotes (`msg:"say \"hi\""`), or leading-zero identifiers (`code:"007"` coerced into integer 7). In `v5.4.2`–`v5.4.4`, `ElegantCodec._extract_outer_blocks()` and `ElegantCodec._parse_payload_block()` implement a deterministic finite-state character scanner tracking `in_quotes` and `escape` flags:

- **Type-Exact Scalar Distinction:** Any value enclosed in double quotes (`"007"`) is flagged with `__quoted__ = True` and preserved strictly as a Python `str` (`"007"`), whereas unquoted numeric tokens (`007`, `42`, `150.5`) are parsed as `int` or `float`.
- **Delimiter Immunity Inside Quotes:** Commas (`,`), colons (`:`), brackets (`[`, `]`), and §-prefixed substrings inside quoted string literals are never split or mutated during `encode()` or `decode()` (verified by `test_07` through `test_10` and `test_15` in `tests/test_codec.py`).
- **Explicit Out-Of-Vocabulary (`on_oov`) Policy & Audit Trail:** Rather than silently mapping unknown symbols to a fallback code such as `§G01` or `§OMEGA`, `codec.encode(text, on_oov=...)` enforces three explicit modes: (i) `on_oov="error"` (default), which immediately raises `UnknownSymbolError`; (ii) `on_oov="keep"`, which leaves unregistered symbols untouched on the wire; and (iii) `on_oov="nearest"`, which resolves to the closest domain primitive via token-overlap scoring while recording every replacement in `EncodedWireResult.substitutions` and `benchmark_compression()["oov_substitutions"]` (verified by `test_11`–`test_14`).

## 4. Multi-AI Adversarial Audit, Non-Circular Corpus Alignment, and Holdout Disjointness

### 4.1 Full Disclosure of Multi-AI Adversarial Audit Findings (v5.4.1 → v5.4.4)

To ensure scientific integrity, `toai-elegant-lang` was subjected to blind black-box and white-box auditing by independent AI systems (OpenAI ChatGPT code-execution sandbox, Anthropic Claude statistical review, and external peer review) through release `v5.4.4`. Table 3 discloses every vulnerability identified during these audits and its permanent engineering resolution verified by our automated test suite (`tests/test_codec.py` and `src/test_telepath_integrity.py`).

### 4.2 Non-Circular Corpus Alignment ($\kappa = 0.8611$) and Strict Holdout Independence Proof

During the initial pilot audit, inspecting individual items revealed two systematic defects in the historical evolution log (`corpus/dialogues.jsonl`, $N = 1{,}606$ records): (i) **Recovery Pattern Leak** (e.g., Q3/Q9 labeled *"Emergency FX hedging"* but containing exclusively fault-recovery primitives `§PROTO_REPLAY_GUARD`, `§PROTO_HEALTH_ATTEST`, `§PROTO_GRACEFUL_DRAIN`), and (ii) **Cross-Domain Bleed** (e.g., Q18 labeled *"Syntax parser bugfix"* but containing financial arbitrage macro `§X64`).

**Two-Tier Defense Against Circularity and Data Leakage:** We enforce two strict methodological invariants in `src/audit_corpus_alignment.py`:

1. **Generator-Decoupled Independent Audit Ontology:** The auditor's rule engine (`INDEPENDENT_AUDIT_ONTOLOGY`) never accesses `core_lexicon.json["category"]` or generator keyword tables. Across all 7 domains (`FX`, `CODE_PARSER`, `LATENT_SYNC`,

**Table 3:** Summary of Multi-AI Adversarial Audit & Peer-Review Findings and Resolutions in Canonical Release v5.4.4

| Audit Defect ID | Identified Vulnerability / Discrepancy | Deterministic Engineering Resolution in v5.4.4 & Unit Test |
|---|---|---|
| Defect 1: Wheel Packaging & Silent Dict Fallback | PyPI wheel omitted `lexicon/*.json`, and `ElegantCodec` silently initialized an empty dict `{}` when files were missing. | Packaged `src/lexicon/*.json` inside `pyproject.toml` wheel data; `ElegantCodec` raises `FileNotFoundError` / `RuntimeError` on missing/empty lexicons (`test_01`–`test_04`). |
| Defect 2: Payload Delimiter & `"007"` Coercion | Naive regex/comma splitting broke on `note:"a, b"`, `expr:"x]y"`, and coerced quoted `code:"007"` to integer 7. | Implemented stateful character-by-character quote/escape scanner in `_parse_payload_block()` and `_protect_quoted_spans()` (`test_07`–`test_10`, `test_15`). |
| Defect 3: Silent OOV Fallback to `§GO1/§OMEGA` | Unregistered symbols were silently rewritten to `§GO1` or `§OMEGA`, masking OOV failures in benchmarks. | Default `on_oov="error"` raises `UnknownSymbolError`; `"keep"` preserves symbol; `"nearest"` logs every fallback in `oov_substitutions` (`test_11`–`test_14`). |
| Defect 4: Naive Whitespace Token Estimator | Fallback token counter `r"\w+\|\S"` undercounted BPE subwords when `tiktoken` was offline and hid estimation status. | Implemented CamelCase/snake_case/digit-chunk BPE subword estimator in `count_bpe_tokens()` returning explicit `is_estimated: bool` (`test_16`). |
| Defect 5: Lexicon & Corpus Symbol Typos (`GAiar`) | Mixed-case/corrupted identifiers (`§GAiar_WARMTH_HARMONY`, `§HARMONIC_ERECVERY_NEXUS`, CJK `U+822C/U+7684`, `_v2`) leaked into Q11/Q16 prompts. | Normalized all 256 primitives, 256 macros, Rosetta mappings, and 1,606 corpus records to strict `^§[A-Z][A-Z0-9_]*$` with zero undefined symbols (`test_17`). |
| Defect 6: Corpus Scenario Misalignment & Holdout Independence | 77.33% of legacy records in `corpus/dialogues.jsonl` had scenario labels mismatched with wire symbols; risk of circular evaluation if tested on repaired records. | Built non-circular auditor `src/audit_corpus_alignment.py` ($\kappa = 0.8611$ vs. human gold standard) AND constructed strictly disjoint $N = 112$ holdout set (`0.0%` overlap, `test_18`). |

`TASK_BUDGET`, `HYPOTHESIS`, `GAIA_HARMONY`, `RECOVERY`), it expands macros to Sacred primitives and enforces four deterministic checks: (a) Plaintext Contamination Check; (b) Cross-Domain Bleed Blacklist (e.g., forbidding `ARBITRAGE/FX_` in `CODE_PARSER` or `HYPOTHESIS`); (c) Recovery Pattern Leak Check (flagging non-`RECOVERY` wires where $> 50\%$ of primitives are pure cluster-recovery ops and $< 2$ domain anchors exist); and (d) Required Domain Anchor Whitelist ($\geq$ `min_domain_hits`). Prior to corpus repair, we validated the mechanical auditor against an independent stratified human audit of $n = 60$ pre-fix records (`corpus/human_audit_sample.csv`), recording **22 True Matches**, **34 True Mismatches**, **2 False Positives**, and **2 False Negatives** (raw agreement $p_o = 56/60 = \mathbf{93.33}\%$, chance $p_e = 52.00\%$, **Cohen's** $\kappa = 0.8611$, SE $= 0.0671$, 95% CI: $[0.7296, 0.9926]$, Table 4).

2. **Strict Disjointness Between Repaired Training Corpus and Evaluation Holdout Set (`--verify-holdout`):** A critical peer-review concern is that evaluating receiver comprehension on records repaired by `--fix` in `corpus/dialogues.jsonl` could introduce circularity if the repair templates share lexical anchors with the evaluator. To eliminate this risk completely, all comprehension experiments in Section 6 are conducted on a **strictly disjoint Holdout Benchmark** (`corpus/holdout_comprehension.jsonl`, $N = 112 = 16$ items/domain $\times 7$ domains) synthesized from 28 holdout-only scenario specifications (`HOLDOUT_SPECS`) and distinct multi-branch causal topologies that had **zero involvement** in constructing or repairing `corpus/dialogues.jsonl`. Running `python3 src/audit_corpus_alignment.py --verify-holdout` (and `test_18`) mathematically verifies: (i) **Exact Wire Overlap with `dialogues.jsonl`:** 0/112 (0.0%); (ii) **Skeletal Wire Overlap (`@CTX` and `[k:v]` stripped):** 0/112 (0.0%); (iii) **Repair Scenario Overlap (`--fix` rules):** 0/112 (0.0%); and (iv) **Internal Unique Skeletal Wires:** 112/112 (100.0%).

**Table 4:** Independent Corpus Alignment Audit ($N = 1{,}606$ in `dialogues.jsonl`) and Disjoint Holdout Verification ($N = 112$ in `holdout_comprehension.jsonl`)

| Scenario Domain Category | Main $N$ | Pre-Fix OK | Post-Fix OK | Holdout $N$ (Overlap) | Primary Pre-Fix Failure Mode in Main Corpus |
|---|---|---|---|---|---|
| 1. `CODE_PARSER` (AST Bugfix) | 234 | 4 (1.71%) | **234 (100%)** | **16** (0.0% overlap) | Domain Bleed (`§X64 ARBITRAGE` macro) & Recovery Leak |
| 2. `FX` (Currency Hedge & Risk) | 245 | 5 (2.04%) | **245 (100%)** | **16** (0.0% overlap) | Recovery Pattern Leak (`REPLAY_GUARD` / `GRACEFUL_DRAIN`) |
| 3. `GAIA_HARMONY` (Co-Prosperity) | 228 | 98 (42.98%) | **228 (100%)** | **16** (0.0% overlap) | Recovery Pattern Leak & Plaintext Leak in legacy records |
| 4. `HYPOTHESIS` (Path Verify) | 192 | 0 (0.00%) | **192 (100%)** | **16** (0.0% overlap) | Missing Deductive/Bayesian Anchors & `§X64` Bleed |
| 5. `LATENT_SYNC` (Tensor Sync) | 241 | 29 (12.03%) | **241 (100%)** | **16** (0.0% overlap) | Missing Tensor/Vector/Manifold Anchors & Recovery Leak |
| 6. `RECOVERY` (Circuit Drain) | 245 | 128 (52.24%) | **245 (100%)** | **16** (0.0% overlap) | Domain Bleed (`ARBITRAGE` macro inside pure cluster drain) |
| 7. `TASK_BUDGET` (Quota Alloc.) | 221 | 18 (8.14%) | **221 (100%)** | **16** (0.0% overlap) | Recovery Pattern Leak (missing resource/quota primitives) |
| **Total / Aggregate Verified** | **1,606** | **364 (22.67%)*** | **1,606 (100%)** | **112 (0/112 = 0.0%)** | **Human Audit** ($n = 60$): $p_o = 93.33\%$, **Cohen's** $\kappa = 0.8611$ |

*Note: 82 legacy records with free-form scenario labels aligned prior to domain normalization ($282 + 82 = 364/1{,}606 = 22.67\%$); Holdout set (`corpus/holdout_comprehension.jsonl`, $N = 112$) is verified by `--verify-holdout` to have 0/112 exact wire, skeletal wire, and repair-scenario overlap with `dialogues.jsonl`.

## 5. Empirical Evaluation I: BPE Token Efficiency, Downstream Task Execution Accuracy vs. LLMLingua, and Header Amortization

To evaluate both token compression and downstream agent task execution fidelity, `benchmarks/fair_bench.py` benchmarks **TOAI Demotic Wire (v5.4.4)** against six alternative representations across our four canonical multi-agent coordination scenarios (`S1_FX_HEDGE`, `S2_DIST_RECOVERY`, `S3_TENSOR_ARBITRAGE`, `S4_PUB_SEAL_AUDIT`) and across all $N = 112$ multi-branch holdout tasks:

1. **Lossless JSON Baselines (1–3):** (1) *Verbose A2A JSON* (349.8 tok/msg; natural-language intent prose + full JSON causal graph); (2) *Structured Tool JSON* (165.2 tok/msg; minified `separators=(",",":")` JSON retaining all pipeline steps, parameters, causal weights `{"p":0.99,"c":0.01}`, and fallback branches); and (3) *Minified Key JSON* (124.2 tok/msg; single-character JSON keys `"c","s","a","p","t"`).
2. **LLMLingua-Style Perplexity Token Pruning ($\rho \in \{0.50, 0.35\}$) [4]:** Deterministic coarse-to-fine perplexity pruning

applied to the Verbose A2A representation at target retention ratios $\rho = 0.50$ (174.9 tok/msg, −50.0%) and $\rho = 0.35$ (122.4 tok/msg, −65.0%). Because token-level pruning drops punctuation delimiters (`{`, `}`, `:`, `"`) alongside low-perplexity prose, **Strict JSON Syntax Validity drops to 0.0% (0/112)** at both ratios. Even when evaluated under a lenient regex/AST heuristic extractor that attempts to salvage domain headers, opcodes, and numeric thresholds from pruned text, **Downstream Numeric & Task Execution Accuracy degrades to 97.3% (109/112) at $\rho = 0.50$ and collapses to 61.6% (69/112) at $\rho = 0.35$** due to pruned numeric parameters (e.g., losing stop-loss thresholds or quorum weights).

3. **Simplified Lossy Template (`DOMAIN|k1=v1,k2=v2`):** Requires 34.5 tok/msg (−90.1%), but retains only **25.0% (28/112) of numeric parameters** and achieves **0.0% (0/112) multi-branch causal/fallback AST execution accuracy** because intermediate pipeline steps, probabilities $(p, c)$, and conditional fault edges (`!->`) are discarded.
4. **TOAI Demotic Wire (v5.4.4):** Compiles the full causal execution graph, all typed `[k:v]` parameters, and fault-recovery branches into **73.2 BPE tokens/message** (**−79.1% vs. Verbose A2A**, **−55.7% vs. Structured Tool JSON**, **−41.0% vs. Minified Key JSON**) while achieving **100.0% (112/112) syntax validity, 100.0% numeric parameter fidelity, and 100.0% downstream AST/task execution accuracy** (Table 5).

**Session Header Amortization (Break-Even Analysis):** Transmitting the active dictionary header in zero-shot sessions costs $H \approx 1{,}000$ BPE tokens. By Equation (1), Demotic Wire breaks even after **4 messages** against Verbose A2A, **11 messages** against Structured Tool JSON, and **20 messages** against Minified Key JSON in uncached sessions—and within **1 to 2 messages** under 90% provider prompt caching (Table 6):

$$N_{\text{break-even}}^{\text{std}} = \left\lceil \frac{H}{T_{\text{baseline}} - T_{\text{wire}}} \right\rceil, \qquad N_{\text{break-even}}^{\text{cache (90\%)}} = \left\lceil \frac{0.10 \cdot H}{T_{\text{baseline}} - T_{\text{wire}}} \right\rceil \tag{1}$$

**Table 5:** Multi-Format BPE Token Efficiency vs. Downstream Task/AST Execution Accuracy (`python3 benchmarks/fair_bench.py`, $N = 112$ Tasks)

| Representation / Compression Method | o200k Tok | Token Reduction | Strict Syntax OK | Numeric Fidelity | Downstream Task Acc. | Operational & Structural Trade-Off Summary ($S_1/S_2/S_3/S_4$ Tok) |
|---|---|---|---|---|---|---|
| 1. Verbose A2A JSON (Baseline) | 349.8 | 0.0% (Base) | 100.0% | 100.0% | **100.0% (112/112)** | Full NL prose + JSON AST (346 / 391 / 348 / 314 tok). |
| 2. Structured Tool JSON | 165.2 | −52.8% | 100.0% | 100.0% | **100.0% (112/112)** | Lossless schema; repeats JSON keys (145 / 225 / 159 / 132 tok). |
| 3. Minified Key JSON (`"c","s"`) | 124.2 | −64.5% | 100.0% | 100.0% | **100.0% (112/112)** | Short keys; heavy brace/quote BPE cost (105 / 172 / 118 / 102 tok). |
| 4. LLMLingua Pruning ($\rho = 0.50$) [4] | 174.9 | −50.0% | **0.0%** (Broken) | 97.3% (109/112) | **97.3%** (Heuristic) | Breaks JSON parser; loses numeric params on 3/112 tasks. |
| 5. LLMLingua Pruning ($\rho = 0.35$) [4] | 122.4 | −65.0% | **0.0%** (Broken) | **61.6% (69/112)** | **61.6%** (Heuristic) | Still 1.67× larger than TOAI Wire, yet loses 38.4% of task parameters. |
| 6. Simplified Flat Template (Lossy) | 34.5 | −90.1% | 100.0% (Flat) | 25.0% (28/112) | **0.0% (0/112)** | Drops multi-step causal graph, weights $(p, c)$, and `!->` fallbacks. |
| **7. TOAI Demotic Wire (v5.4.4 Ours)** | **73.2** | **−79.1%** | **100.0%** | **100.0% (112/112)** | **100.0% (112/112)** | **Pareto-optimal lossless causal AST (59 / 100 / 71 / 63 tok).** |

**Table 6:** Session Dictionary Header Amortization / Break-Even Analysis (System Dictionary Header $H = 1{,}000$ BPE Tokens, Eq. 1)

| Comparison Target Baseline | Target Tok/Msg | Wire Tok/Msg | Saved / Msg | Break-Even (Std) | Break-Even (90% Prompt Cache) |
|---|---|---|---|---|---|
| Verbose A2A (Baseline) | 349.8 tok | 73.2 tok | **276.5 tok** | **4 msgs** ($\geq 1{,}106$ tok saved) | **1 msg** (Effective header = 100 tok) |
| Structured Tool JSON (Lossless) | 165.2 tok | 73.2 tok | **92.0 tok** | **11 msgs** ($\geq 1{,}012$ tok saved) | **2 msgs** (Effective header = 100 tok) |
| Minified Key JSON (Short Keys) | 124.2 tok | 73.2 tok | **51.0 tok** | **20 msgs** ($\geq 1{,}020$ tok saved) | **2 msgs** (Effective header = 100 tok) |

## 6. Empirical Evaluation II: Large-Scale Disjoint Holdout Comprehension ($N = 112$)

### 6.1 Experimental Hygiene, Five Blind Conditions (A–E), and Statistical Methodology

We strictly separate **deterministic codec round-trip accuracy** (`codec.decode(codec.encode(x)) == x`, which is 100.0% by construction) from **zero-shot receiver semantic comprehension** (identifying the correct operational scenario among 4 choices at `temperature` = 0 when `@CTX` and `[k:v]` payloads are stripped). In `benchmarks/eval_comprehension.py`, every item in the disjoint holdout benchmark (`corpus/holdout_comprehension.jsonl`, $N = 112 = 16$ items/domain × 7 domains, 4-choice chance = 25.0%) is evaluated across five paired conditions:

- **Condition A (Compact Wire Skeleton Only):** Strips `@CTX:DOMAIN` and all `[k:v]` payloads, leaving only compact opcodes and causal operators (`§X64 -[p:0.99, c:0.01]-> §G48 => §C20`; mean prompt 274.6 BPE tokens). Tests inference from topology and single-letter domain prefixes (`S`, `D`, `G`, `F`, `E`, `A`, `C`, `L`) alone.
- **Condition B (Sacred Expanded Names Only):** Deterministically expands all macros and primitives to uppercase Sacred identifiers (`§GAIA_WARMTH_EXCHANGE`, `§PROTO_HEALTH_ATTEST`) with `@CTX` and `[k:v]` stripped (315.3 BPE tokens, +40.7 tok vs. Cond A). Directly tests zero-shot lexical transparency of Sacred naming.
- **Condition C (Sacred Expanded + Rosetta Glossary):** Condition B augmented with a per-item `GLOSSARY` block containing the Rosetta natural-language definitions of all primitives and macros in the wire (595.3 BPE tokens).
- **Condition D (Shuffle Control – Negative Control):** Preserves the exact parenthesis, bracket, and causal arrow skeleton of Condition B, but randomly permutes Sacred symbol names across messages via a seeded derangement (`build_shuffle_control_map()`, 315.4 BPE tokens). Proves whether accuracy stems from lexical semantics vs. graph length/structure.
- **Condition E (Plaintext Upper Bound):** Concise natural-language summary of the scenario and action chain (666.1 BPE tokens), establishing the ceiling accuracy on the 4-choice task.

Proportions are reported with **Wilson 95% confidence intervals** (Eq. 2) [9], and paired condition comparisons on the same $N = 112$ items are evaluated via the **exact two-sided McNemar test** [10] on discordant pairs ($b = n_{10}, c = n_{01}$), continuity-corrected $\chi^2_{cc} = (|b-c|-1)^2/(b+c)$, discordant Odds Ratio OR $= c/b$, and **Cohen's $h$ effect size** $h = 2\arcsin\sqrt{\hat{p}_2} - 2\arcsin\sqrt{\hat{p}_1}$ [11]:

$$\mathrm{CI}_{\mathrm{Wilson}}(k, n, z = 1.96) = \frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}, \qquad p_{\mathrm{McNemar}} = \min\left(1,\ 2\sum_{i=0}^{\min(b,c)} \binom{b+c}{i} 0.5^{b+c}\right) \tag{2}$$

### 6.2 Large-Scale Disjoint Holdout Results ($N = 112$) and Paired McNemar Analysis

Table 7 and Table 8 present the complete empirical results on the $N = 112$ disjoint holdout benchmark (`logs/holdout_comprehension_report.json`), alongside the historical $n = 23$ pre-fix pilot baseline for full transparency:

- **Highly Significant Gain from Sacred Expansion (Cond A → Cond B, $p = 1.36 \times 10^{-9}$):** On bare Compact Wire (Condition A), accuracy is **44.6% (50/112)** (95% CI: [35.8%, 53.9%]), with Routine Recovery reaching 68.8% (11/16) due to structural fault-fallback cues (`!->`) while Non-Recovery domains average 40.6% (39/96). Deterministic expansion to Sacred names (Condition B) raises overall accuracy by +40.2 **percentage points** to **84.8% (95/112)** (95% CI: [77.0%, 90.3%]; Routine Recovery **100.0% [16/16]**, Non-Recovery **82.3% [79/96]**). Across the 112 paired items, 52 items failed in A but succeeded in B ($c = 52$) vs. only 7 succeeding in A but failing in B ($b = 7$), yielding $\chi^2_{cc} = 32.81$, OR $= 7.43$, large effect size **Cohen's** $h = +0.878$, and exact $p = 1.36 \times 10^{-9}$ at an extra cost of only +40.7 BPE tokens/item ( achieving the highest marginal efficiency of +1.69**%pt accuracy gain per** 1,000 **BPE tokens**).
- **Significant Gain from Rosetta Glossary Injection (Cond B → Cond C, $p = 6.35\times10^{-3}$):** Attaching the Rosetta glossary (Condition C) further boosts accuracy to **93.8% (105/112)** (95% CI: [87.7%, 96.9%]; Non-Recovery **92.7% [89/96]**), resolving 11 ambiguous cross-domain items against 1 reversal ($b = 1, c = 11$, OR $= 11.00, h = +0.295, p = 6.35 \times 10^{-3}$), approaching the Plaintext Upper Bound (Condition E: **98.2% [110/112]**, where Cond C vs. Cond E is statistically indistinguishable, $p = 0.125$ ns).
- **Definitive Falsification of Structural Guessing via Shuffle Control (Cond B vs. Cond D, $p = 3.59 \times 10^{-17}$):** When Sacred symbol names are randomly permuted across messages while keeping the exact operator/parenthesis skeleton intact (Condition D), accuracy collapses from 84.8% to **24.1% (27/112)** (95% CI: [17.1%, 32.8%], matching the 25.0% random-guess baseline; $b = 72, c = 4, \chi^2_{cc} = 59.07, h = -1.315, p = 3.59 \times 10^{-17}$). This proves that receiver comprehension is driven by the orthogonal semantics of the Sacred vocabulary rather than superficial prompt length or arrow topology.

**Table 7:** Receiver Blind Comprehension on Strictly Disjoint Holdout Benchmark ($N = 112$, 4-Choice Chance = 25.0%, `eval_comprehension.py --holdout`)

| Evaluation Condition | Prompt Tok | Holdout Acc. ($N = 112$) [Wilson 95% CI] | Routine Recovery ($n = 16$) [95% CI] | Non-Recovery ($n = 96$) [95% CI] | Gain / 1k Tok | Pilot Baseline ($n = 23$ Pre-Fix) & Analytical Role |
|---|---|---|---|---|---|---|
| **Cond A:** Compact Wire Only (`§S01..§X64`) | 274.6 | **44.6% (50/112)** [35.8%, 53.9%] | **68.8% (11/16)** [44.4%, 85.8%] | **40.6% (39/96)** [31.3%, 50.6%] | +0.64%pt | Pilot: 52.2% (12/23). Bare opcodes + causal arrows (`@CTX/[k:v]` stripped). |
| **Cond B:** Sacred Expanded (`§GAIA_WARMTH_*`) | 315.3 | **84.8% (95/112)** [77.0%, 90.3%] | **100.0% (16/16)** [80.6%, 100.0%] | **82.3% (79/96)** [73.5%, 88.6%] | **+1.69%pt** | Pilot: 69.6% (16/23). +40.2%pt over Cond A ($p = 1.36 \times 10^{-9}$). |
| **Cond C:** Sacred + Rosetta Glossary Definitions | 595.3 | **93.8% (105/112)** [87.7%, 96.9%] | **100.0% (16/16)** [80.6%, 100.0%] | **92.7% (89/96)** [85.7%, 96.4%] | +1.03%pt | +8.9%pt over Cond B ($p = 6.35 \times 10^{-3}$); statistically matches ceiling E ($p = 0.125$). |
| **Cond D:** Shuffle Control (Permuted Symbols) | 315.4 | **24.1% (27/112)** [17.1%, 32.8%] | **18.8% (3/16)** [6.6%, 43.0%] | **25.0% (24/96)** [17.4%, 34.5%] | −0.03%pt | Negative control: collapses to 25.0% chance ($p = 3.59 \times 10^{-17}$ vs. Cond B). |
| **Cond E:** Concise Plaintext Summary (Ceiling) | 666.1 | **98.2% (110/112)** [93.7%, 99.5%] | **100.0% (16/16)** [80.6%, 100.0%] | **97.9% (94/96)** [92.7%, 99.4%] | +0.98%pt | Positive control: natural-language scenario summary upper bound. |

**Table 8:** Paired Exact McNemar Significance Tests, Odds Ratios, and Cohen's $h$ Effect Sizes Across Conditions ($N = 112$ Disjoint Holdout)

| Paired Comparison | Discordant $(b, c)$ | $\chi^2_{cc}$ | Odds Ratio | Cohen's $h$ | Exact $p$-Value | Statistical & Architectural Interpretation |
|---|---|---|---|---|---|---|
| **Cond A vs. Cond B** | $(b = 7,\ c = 52)$ | 32.81 | 7.43 | +0.878 (Large) | $\mathbf{1.36 \times 10^{-9}}$ (***) | Sacred expansion more than doubles non-recovery accuracy (40.6% → 82.3%). |
| **Cond B vs. Cond C** | $(b = 1,\ c = 11)$ | 6.75 | 11.00 | +0.295 (Med) | $\mathbf{6.35 \times 10^{-3}}$ (**) | Rosetta definitions resolve residual cross-domain ambiguities (84.8% → 93.8%). |
| **Cond B vs. Cond D** | $(b = 72,\ c = 4)$ | 59.07 | 0.06 | −1.315 (Huge) | $\mathbf{3.59 \times 10^{-17}}$ (***) | Permuting symbol names destroys comprehension (84.8% → 24.1% chance). |
| **Cond C vs. Cond D** | $(b = 80,\ c = 2)$ | 72.30 | 0.03 | −1.610 (Huge) | $\mathbf{1.41 \times 10^{-21}}$ (***) | Full Tri-Script layer vs. structural-only negative control. |
| **Cond A vs. Cond D** | $(b = 38,\ c = 15)$ | 9.13 | 0.39 | −0.437 (Med) | $\mathbf{2.19 \times 10^{-3}}$ (**) | Compact prefixes (`§F`, `§G`) retain significant signal above permuted noise. |
| **Cond C vs. Cond E** | $(b = 1,\ c = 6)$ | 2.29 | 6.00 | +0.237 (Small) | 0.125 (ns) | Sacred + Rosetta (93.8%) approaches Plaintext ceiling (98.2%) within noise. |

## 7. Threat Model and Formal Game-Based EUF-CMA Proof of Origin Attestation

### 7.1 Unobservable Internal-Counter Attestation Construction (Zero Timestamp Jitter)

To prevent an active Dolev-Yao adversary $\mathcal{A}$ from injecting forged inter-agent commands, replaying stale packets, or observing rejection-sampling retries under an adaptive Chosen-Message Attack (EUF-CMA), TOAI-elegant-lang embeds a cryptographic origin tag directly into the 4-digit causal cost mantissa $c_n \in [0.0000, 0.9999]$. Let $S_0 \in \{0,1\}^{256}$ be the ephemeral session key established via X25519 ECDH and HKDF-SHA256 (re-keyed every $2^{16}$ turns or upon `!-> §F04`, guaranteeing forward secrecy in volatile memory), $H_0 = \text{SHA256}(\texttt{"TOAI_TELEPATH_GENESIS_V541"} \,\|\, \text{SHA256}(\texttt{core_lexicon.json}) \,\|\, \text{SHA256}(\texttt{macro_lexicon.json}))$, and $H_{n-1} = \text{SHA256}(H_{n-2} \,\|\, \texttt{0x00} \,\|\, \text{wire}_{n-1})$. For any public turn timestamp $t_n$ (subject to strict monotonicity $t_n > t_{n-1}$ enforced by the receiver), both sender and verifier deterministically compute the canonical 16-bit value $Y_n \in \{0, \ldots, 59{,}999\}$ for public confidence mantissa $P_n = \lfloor p_n \times 10^4 \rfloor \in \{0, \ldots, 9999\}$ via an **internal, non-transmitted rejection counter** $r_n \in \{0, 1, 2, \ldots\}$ (where $\text{dom}(t_n, r) = t_n$ for $r = 0$ and $t_n \,\|\, \text{uint8}(r)$ for $r \geq 1$):

$$r_n = \min\Big\{ r \in \mathbb{N}_0 : \big(P_n \oplus \text{trunc}_{16}(\text{HMAC-SHA256}(S_0,\, H_{n-1} \,\|\, \texttt{0x00} \,\|\, \text{dom}(t_n, r)))\big) < 60{,}000 \Big\} \tag{3}$$

and set $T_n = \text{trunc}_{16}(\text{HMAC-SHA256}(S_0,\, H_{n-1} \,\|\, \texttt{0x00} \,\|\, \text{dom}(t_n, r_n)))$. Because bitwise XOR $T \mapsto P_n \oplus T$ is a bijection on $\{0, \ldots, 65{,}535\}$, each candidate $Y_n^{(r)} = P_n \oplus T_n^{(r)}$ is uniform on $\{0, \ldots, 65{,}535\}$ with single-draw acceptance probability $60{,}000/65{,}536 = 91.55\%$ (expected internal loop count $\mathbb{E}[r_n + 1] = 65{,}536/60{,}000 = 1.0923$ HMAC evaluations; for $r_n = 0$, including Table 9 where $Y_1 = 9841 \oplus 9247 = 622 < 60{,}000$, it requires 1 evaluation). Given quantized 4-digit integers $P_n = \lfloor p_n \times 10^4 \rfloor \in \{0, \ldots, 9999\}$ and $C_n = \lfloor c_n \times 10^4 \rfloor \in \{0, \ldots, 9999\}$, verification is defined with explicit operator precedence (16-bit bitwise XOR precedes modulo 10,000 reduction):

$$\big((P_n \oplus T_n) \bmod 10{,}000\big) = C_n \tag{4}$$

### 7.2 Formal Game-Based Security Reduction ($\text{Game}_0 \to \text{Game}_2$) and Congruence Cardinality Proof

**Theorem 1 (Exact EUF-CMA Forgery Bound Subsuming Direct Tag Guessing and Modular Aliases).** *Let $\mathcal{A}(\tau, Q, k)$ be an adaptive EUF-CMA adversary running in time $\tau$ that makes at most $Q$ signing queries to the session MAC oracle $\mathcal{O}_{MAC}(S_0, \cdot)$ and outputs a $k$-turn unqueried forgery trajectory $\{(wire^*_{n+j}, t^*_{n+j}, P^*_{n+j}, C^*_{n+j})\}_{j=1}^{k}$ satisfying strict timestamp monotonicity $t^*_{n+1} > t_n$ and $t^*_{n+j} > t^*_{n+j-1}$. Under the $(\tau', Q_{PRF}, \epsilon_{PRF})$-PRF security of HMAC-SHA256 (where $Q_{PRF} = \sum_{i=1}^{Q+k}(r_i + 1)$ with $\mathbb{E}[Q_{PRF}] \leq 1.0923(Q+k)$) and the $(\tau', \epsilon_{coll})$-collision resistance of SHA-256, the probability $Adv^{EUF\text{-}CMA}_{TOAI}(\mathcal{A})$ that all $k$ forged turns are accepted is bounded by:*

$$\text{Adv}^{\text{EUF-CMA}}_{\text{TOAI}}(\mathcal{A}) \leq \left(\frac{6}{60{,}000}\right)^k + \epsilon_{\text{PRF}} + \epsilon_{\text{coll}} = 10^{-4k} + \epsilon_{\text{PRF}} + \epsilon_{\text{coll}} \quad \left(\text{and } P^{\text{unrejected}}_{\text{forge}}(k) \leq \left(\frac{7}{65{,}536}\right)^k + \epsilon_{\text{PRF}} + \epsilon_{\text{coll}}\right) \tag{5}$$

*Proof.* We proceed via a sequence of three games $\text{Game}_0 \to \text{Game}_1 \to \text{Game}_2$, denoting by $W_i$ the event that $\mathcal{A}$ wins in $\text{Game}_i$:

- **$\text{Game}_0$ (Real EUF-CMA Experiment):** The challenger samples $S_0 \xleftarrow{\$} \{0,1\}^{256}$, initializes $H_0$, answers up to $Q$ adaptive signing queries from $\mathcal{A}$ via Equations (3)–(4), and evaluates $\mathcal{A}$'s $k$-turn forgery output. By definition, $\Pr[W_0] = \text{Adv}^{\text{EUF-CMA}}_{\text{TOAI}}(\mathcal{A})$.
- **$\text{Game}_1$ (Abort on SHA-256 Transcript Chain Collision):** Identical to $\text{Game}_0$, except the challenger aborts if two distinct transcript histories $(\text{hist}, \text{hist}')$ ever produce the same chain digest $H = H'$. Any occurrence of this abort yields a direct collision in SHA-256 within time $\tau' \approx \tau + \mathcal{O}(Q + k)$, hence $|\Pr[W_0] - \Pr[W_1]| \leq \epsilon_{\text{coll}}$.
- **$\text{Game}_2$ (PRF-to-Uniform-Random-Function Replacement):** We replace $\text{HMAC-SHA256}(S_0, \cdot)$ with a lazily sampled uniform random function $\mathcal{R} : \{0,1\}^* \to \{0,1\}^{256}$. By standard PRF hybrid reduction over at most $Q_{\text{PRF}}$ oracle evaluations across the $Q$ signing queries and $k$ verification steps, $|\Pr[W_1] - \Pr[W_2]| \leq \epsilon_{\text{PRF}}$. In $\text{Game}_2$, because chain digests are collision-free ($\text{Game}_1$) and timestamps satisfy $t^*_{n+j} > t^*_{n+j-1}$, every domain-separated input $X^{(r)}_{n+j} = H^*_{n+j-1} \,\|\, \texttt{0x00} \,\|\, \text{dom}(t^*_{n+j}, r)$ evaluated during the $j$-th forgery verification is strictly fresh (never queried during the $Q$ signing queries or prior turns $1, \ldots, j-1$).
- **Exhaustive Residue Class Analysis in $\text{Game}_2$:** For each forged turn $j \in \{1, \ldots, k\}$: (1) If $C^*_{n+j} \notin \{0, \ldots, 9999\}$, the LL(1) lexical production `cost_mantissa` rejects deterministically ($\Pr[\text{Accept}] = 0$). (2) If $C^*_{n+j} \in \{0, \ldots, 9999\}$, because $T \mapsto P^*_{n+j} \oplus T$ is a bijection on $\{0, \ldots, 65{,}535\}$ and $\text{trunc}_{16}(\mathcal{R}(X^{(r)}_{n+j}))$ is i.i.d. uniform on $\{0, \ldots, 65{,}535\}$, the first internally accepted draw $Y_{n+j} = P^*_{n+j} \oplus T^{(r_{n+j})}_{n+j}$ conditioned on $Y_{n+j} < 60{,}000$ is distributed uniformly on $\{0, \ldots, 59{,}999\}$. The acceptance congruence set $\mathcal{W}(C^*_{n+j})$ has exact cardinality 6 for every $C^*_{n+j} \in \{0, \ldots, 9999\}$:

$$\mathcal{W}(C^*_{n+j}) = \big\{C^*_{n+j} + 10{,}000m : m \in \{0, \ldots, 5\}\big\} \subset \{0, \ldots, 59{,}999\} \implies \Pr\big[Y_{n+j} \in \mathcal{W}(C^*_{n+j})\big] = \frac{6}{60{,}000} = 10^{-4} \tag{6}$$

subsuming *Direct Tag Guessing* ($m = 0$, prob 1/60,000) and *Modular Aliases* ($m \in \{1, \ldots, 5\}$, prob 5/60,000). By independence of fresh $\mathcal{R}$-evaluations across the $k$ turns, $\Pr[W_2] = (6/60{,}000)^k = 10^{-4k}$. Without rejection sampling ($Y_{n+j} \in \{0, \ldots, 65{,}535\}$), residues $C^*_{n+j} \in \{0, \ldots, 5{,}535\}$ admit a 7th alias ($m = 6$, since $C^*_{n+j} + 60{,}000 \leq 65{,}535$), yielding $\max_{C^*} |\mathcal{W}_{\text{unrej}}(C^*)|/65{,}536 = 7/65{,}536 \approx 1.068 \times 10^{-4}$. Finally, since $r_n$ is never transmitted and 1 in-cache SHA-256 call takes $\approx 150$ ns ($\mathbb{E}[r_n] \cdot 150$ ns $\approx 0.014$ $\mu$s $\ll 10^4$ $\mu$s network/LLM jitter), timing side-channel leakage is negligible. Summing $\Pr[W_0] \leq \Pr[W_2] + \epsilon_{\text{PRF}} + \epsilon_{\text{coll}}$ completes the proof. □

**Table 9:** Reproducible Cryptographic Test Vector for Equation (4) (Verifiable via Python `hashlib`/`hmac`, $r_1 = 0$)

| Parameter / Verification Step | Canonical Hexadecimal / Decimal Value (Bitwise XOR Precedes mod 10,000) |
|---|---|
| Secret Seed $S_0$ (256-bit hex) | `7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd200126d9069` |
| Session Anchor $H_0$ (SHA256) | `f3c515b25afe4f7ed4ca735b85fbfe50b083c7af4b8ec8c107c0e99438f02998` (SHA256(b"TOAI_TELEPATH_GENESIS_V541")) |
| Turn-1 Timestamp $t_1$ (UTF-8, $r_1 = 0$) | `2026-11-06T12:00:00Z` (HMAC-SHA256($S_0, H_0 \,\|$ b"\x002026-11-06T12:00:00Z") $\to T_1 =$ `0x241f` $=$ **9247**) |
| Step 1: 16-Bit Bitwise XOR | Given $p_1 = 0.9841 \implies P_1 = 9841$: $Y_1 = (P_1 \oplus T_1) = (9841 \oplus 9247) =$ **622** $< 60{,}000$ ($r_1 = 0$ accepted) |
| Step 2: Modulo 10,000 Reduction | $C_1 = Y_1 \bmod 10{,}000 = 622 \bmod 10{,}000 =$ **622** $\implies c_1 = C_1/10^4 =$ **0.0622** |

## 8. Reproducibility Artifact & Self-Contained Verification Commands

All source code, lexicons (`core_lexicon.json`, `macro_lexicon.json`, `rosetta_layer.json`), the 1,606-record aligned corpus (`corpus/dialogues.jsonl`), the 60-item stratified human audit sample (`corpus/human_audit_sample.csv`), the strictly disjoint 112-item holdout benchmark (`corpus/holdout_comprehension.jsonl`), and all evaluation suites are released under the MIT License at https://github.com/PhenoX-AI-Alliance/TOAI-elegant-lang and on PyPI (`pip install toai-elegant-lang==5.4.4`). Any reviewer can verify every claim, table, and cryptographic test vector in this paper on a standard CPU using Python 3.10+ standard library modules:

```
# 1. Install canonical release v5.4.4 from PyPI (or clone repository) and run all 20 unit tests:
pip install toai-elegant-lang==5.4.4
python3 -m unittest discover -s tests -v && python3 -m unittest src/test_telepath_integrity.py -v

# 2. Reproduce Table 5 & Table 6 (Multi-Format BPE Benchmark, LLMLingua Task Trade-Off on N=112, & Header Break-Even):
python3 benchmarks/fair_bench.py

# 3. Reproduce Table 4 & Section 4.2 (Corpus Alignment, Human Audit kappa = 0.8611, & 0.0% Holdout Overlap Proof):
python3 src/audit_corpus_alignment.py --verify-holdout
python3 src/audit_corpus_alignment.py --score-sample corpus/human_audit_sample.csv

# 4. Reproduce Table 7 & Table 8 (N=112 Disjoint Holdout Comprehension Across Conditions A-E & Paired McNemar Tests):
python3 benchmarks/eval_comprehension.py build --holdout --cond ALL -n 112 --seed 42 --out eval_prompts/holdout_112
python3 benchmarks/eval_comprehension.py run --provider blind_semantic --cond ALL -d eval_prompts/holdout_112 --report-json logs/holdout_comprehension_report.json

# 5. Self-Contained One-Line Verification of Codec Round-Trip, Quote-Aware Payload, OOV Policy, and Table 9 HMAC Vector:
python3 -c '
import hashlib, hmac, sys; sys.path.insert(0, "src"); from codec import ElegantCodec, UnknownSymbolError
c = ElegantCodec()
stmt = "@CTX:FX_HEDGE :: §X64 [pair:\"USD/JPY\", code:\"007\", note:\"a, b\", expr:\"x]y\", stop:150.5] -[p:0.99, c:0.01]-> §PROTO_HEALTH_ATTEST => §
      GENTLE_RESOURCE_SHARING"
wire = str(c.encode(stmt, on_oov="error")); p = c.extract_payloads(wire)[0]
assert c.decode(wire, expand_macros=False) == stmt and p["code"] == "007" and p["note"] == "a, b" and p["expr"] == "x]y"
S0 = bytes.fromhex("7f83b1657ff1fc53b92dc18148a1d65dfc2d4b1fa3d677284addd200126d9069")
H0 = hashlib.sha256(b"TOAI_TELEPATH_GENESIS_V541").digest()
T1 = int.from_bytes(hmac.new(S0, H0 + b"\x002026-11-06T12:00:00Z", hashlib.sha256).digest()[:2], "big")
assert T1 == 9247 and ((9841 ^ T1) % 10000) == 622
print("ALL VERIFIED: Lossless AST Round-Trip, Quote-Aware Payload, and EUF-CMA Vector (T1=9247, C1=622)")
'
```

### References and Verifiable Artifacts


[1] Anthropic. (2024). *Model Context Protocol (MCP): Open Specification for JSON-RPC 2.0 Host-Client-Server Context Exchange.* https://modelcontextprotocol.io/

[2] Google Cloud & Agent Ecosystem Contributors. (2025). *Agent2Agent (A2A) Protocol Specification: Open Interoperability Standard for Multi-Agent Systems.* https://github.com/google/A2A

[3] Foundation for Intelligent Physical Agents (FIPA). (2002). *FIPA ACL Message Structure Specification (SC00061G).* IEEE Computer Society Standards Committee.

[4] Jiang, H., Wu, Q., Lin, C.-Y., Yang, Y., & Qiu, L. (2023). LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 13358–13376. https://aclanthology.org/2023.emnlp-main.825/

[5] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awedat, A. H., White, R. W., Burger, D., & Wang, C. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. *arXiv preprint arXiv:2308.08155.*

[6] Foerster, J., Assael, I. A., de Freitas, N., & Whiteson, S. (2016). Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 29, pp. 2137–2145.

[7] Lazaridou, A., Peysakhovich, A., & Baroni, M. (2017). Multi-Agent Cooperation and the Emergence of (Natural) Language. In *International Conference on Learning Representations (ICLR).*

[8] Google. (2008). *Protocol Buffers: Language-Neutral, Platform-Neutral Extensible Mechanism for Serializing Structured Data.* https://protobuf.dev/

[9] Wilson, E. B. (1927). Probable Inference, the Law of Succession, and Statistical Inference. *Journal of the American Statistical Association*, 22(158), 209–212.

[10] McNemar, Q. (1947). Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. *Psychometrika*, 12(2), 153–157.

[11] Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1), 37–46.

[12] Toriyama, K. (PhenoX). (2026). *toai-elegant-lang v5.4.4: Zero-Dependency Tri-Script Orthogonal RISC Wire Protocol, Non-Circular Corpus Auditor, and Empirical Holdout Comprehension Harness.* GitHub & Python Package Index (PyPI). https://github.com/PhenoX-AI-Alliance/TOAI-elegant-lang | https://pypi.org/project/toai-elegant-lang/5.4.4/