Independent Submission
Internet-Draft
Updates: draft-mackay-aacp-00 (if approved)
Intended status: Informational
Expires: 1 December 2026

A. Mackay
Independent

30 May 2026

Agent Action Compression Protocol (AACP) Version 1.1
draft-mackay-aacp-01

Abstract

This document defines the Agent Action Compression Protocol (AACP), a pipe-delimited coordination format for agent-to-agent communication in multi-agent large language model (LLM) systems. AACP transforms verbose natural language coordination instructions into deterministic, typed, machine-parseable packets that can be validated before transmission, logged as structured audit records, and replayed consistently across workflow runs.

For known workflow types, a rule-based encoder produces AACP packets deterministically at zero LLM cost. A three-tier fallback encoder extends this to novel instructions: exact matches are served from a local registry cache at zero cost; pattern matches route to rule-based encoding at zero cost; novel instructions are encoded via LLM and logged to the registry, ensuring each novel pattern incurs at most one LLM encoding call regardless of how many times it subsequently appears.

As a secondary benefit, AACP reduces coordination token usage by approximately 23 percent versus equivalent natural language instructions, measured against live API tokenisation on Claude Sonnet 4.5 and GPT-4o. Consistent token reduction has been validated across four models: Claude Sonnet 4.5, GPT-4o, GPT-4.1, and GPT-4.1-mini.

AACP operates above transport protocols such as the Model Context Protocol (MCP) and Agent-to-Agent Protocol (A2A), compressing message payload content rather than addressing routing or delivery. The protocol is transport-agnostic, model-agnostic, and designed to complement rather than replace existing agent coordination infrastructure.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

1. Introduction

Multi-agent LLM systems coordinate by exchanging instructions between autonomous agents. In current implementations, these coordination messages are typically written in natural language -- verbose, ambiguous, and expensive to tokenise. More significantly, natural language coordination instructions are non-deterministic: the same intent may be expressed differently across runs, making coordination messages difficult to validate, log, replay, or audit.

AACP addresses both problems. It replaces natural language coordination instructions with a compact, typed, pipe-delimited packet format that is deterministic, schema-validated, and structured for machine parsing.

Consider a four-agent payroll workflow where an orchestrating agent instructs an HRMS agent to retrieve salary records. A typical English instruction:

"Please retrieve the employee salary records for the period ending 31 August 2024. I need all active employees, their departments, cost centres, base salary, any changes made this month, and pension contribution rates. Return as JSON array."

This instruction consumes 56 tokens on Claude Sonnet 4.5. It is verbose, includes pleasantries, and will vary in wording across runs. The equivalent AACP packet:

```
FETCH|HR|return:HR-Agent|p:1|aacp:1.1|res:emp_salary|
period:2024-08|filter:status=active|fmt:json
```

consumes 52 tokens, is produced identically on every run by the rule-based encoder, validates against the v1.1 schema before transmission, and can be logged as a structured audit record without post-processing.

Across a four-hop payroll workflow the total coordination token reduction is 22.9 percent on Claude Sonnet 4.5 and 23.7 percent on GPT-4o, measured from live API usage_metadata. The rule-based encoder produces all four packets at zero LLM cost.

Beyond token reduction, AACP provides:

* Deterministic encoding -- identical input produces identical output

* Zero-cost encoding for known workflows via rule-based encoders

* Schema validation before packet transmission

* Structured, machine-readable audit trail without post-processing

* Self-improving fallback with registry cache

* Model-agnostic format validated across four LLM providers

This document specifies AACP version 1.1 and updates draft-mackay-aacp-00 with validated multi-model results, the closed fallback loop specification, and reference to the working open-source implementation.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Agent: An autonomous software process that receives instructions, performs tasks using one or more LLM API calls, and returns results.

Coordination message: An instruction sent from one agent to another describing what action to take, on what resource, with what parameters, and where to return the result. Distinct from task content.

AACP packet: A pipe-delimited string conforming to the format defined in Section 4, used as a coordination message.

Task tokens: Tokens consumed by an agent performing its actual work. AACP does not reduce task tokens.

Coordination tokens: Tokens consumed by the coordination message itself. AACP reduces these by approximately 23 percent versus verbose English.

Rule-based encoder: A deterministic encoder producing AACP packets from structured input without an LLM call. Zero API cost per encoding.

LLM encoder: An encoder using an LLM API call to compress an English instruction into an AACP packet. Used for novel instructions outside known workflow patterns.

Registry: A local store of previously LLM-encoded instructions and their resulting AACP packets, enabling cache lookup on subsequent identical or similar instructions.

3. Motivation and Problem Statement

The problem of verbosity in agent communication has been identified independently by multiple research groups. Mou et al. (2025) [ECOLANG] observed that "there exists redundancy in current agent communication" and achieved greater than 20 percent token reduction through evolved compression language for social simulation. AACP addresses the same problem with a typed, portable packet schema targeting business workflow coordination.

Beyond verbosity, the non-deterministic nature of natural language coordination creates operational problems at scale. When coordination messages are expressed in free-form English, the same intent produces different messages on different runs. This makes it difficult to validate messages before transmission, correlate messages across runs for audit purposes, or replay workflows reliably.

The specific technical gap AACP fills: neither MCP [MCP] nor A2A [A2A] address the semantic content of coordination messages. Both protocols operate at the transport and routing layers. AACP operates at the content layer, defining what agents say to each other rather than how messages are delivered.

At enterprise scale, coordination token overhead is a measurable cost and sustainability concern. Agentic systems consume 5 to 30 times more tokens per task than standard single-turn interactions. In multi-agent workflows, coordination messages compound with every agent hop. For a payroll system processing 500 monthly runs, the coordination token reduction provided by AACP eliminates the cost of 500 LLM encoding calls and reduces input tokens on every coordination hop.

4. Packet Format

An AACP packet is a sequence of pipe-delimited fields. The pipe character (U+007C, "|") is the field separator.

packet = task "|" dom "|" named-fields

TASK and DOM are positional (fields 0 and 1 respectively). All other fields MUST be named key:value pairs. Empty positional slots MUST NOT appear in v1.1 packets.

4.1. Positional Fields

Field 0 -- TASK: The action verb. REQUIRED. MUST be one of the values defined in Section 5.1.

Field 1 -- DOM: The business domain context. REQUIRED. MUST be one of the values defined in Section 5.2.

4.2. Named Fields

After the two positional fields, all fields are named using the format key:value. The following named fields MUST appear in every packet:

return: The agent identifier that receives the result. REQUIRED.

aacp: The protocol version. REQUIRED. MUST be "1.1" for packets conforming to this specification.

The following named fields are RECOMMENDED where applicable:

p: Priority. 1=critical, 2=medium (default), 3=low.

res: The resource identifier the action applies to.

period: A time period, expressed as YYYY-MM or similar.

filter: A filter expression applied to the resource.

fields: Comma-separated return fields. SHOULD be omitted when the receiving agent has default field definitions in its system configuration.

fmt: The requested response format (e.g. json, pdf, xlsx).

4.3. Extended Fields

Additional named fields MAY be appended after core fields. Defined extended keys include: src, src_prev, rules, validate, tmpl, data_ptr, amt, ccy, sup, match, terms, type, party, clause, issue, risk, block, flags, req, highlight, status, to, subj, att, flag_msg, tone, sentiment, actor, chain, prog, ltv, loyalty, urgency.

Unknown extended keys MUST generate advisory warnings in validators, not errors. This permits forward compatibility and organisation-private extensions.

4.4. Packet Examples

Fetch active employee salary records:

```
FETCH|HR|return:HR-Agent|p:1|aacp:1.1|res:emp_salary|
period:2024-08|filter:status=active|fmt:json
```

Merge datasets and calculate payroll:

```
MERGE|HR|return:HR-Agent|p:1|aacp:1.1|rules:payroll_v2|
validate:budget_cc
```

Flag a legal clause for review:

```
FLAG|LEGAL|return:LEG-Agent|p:1|aacp:1.1|type:NDA|
party:Acme-Ltd|clause:s7|issue:ip_rights_restriction|
risk:high|block:signature
```

Build IT user account:

```
BUILD|IT|return:IT-Agent|p:1|aacp:1.1|res:ad_account|
filter:usr=j.smith|fields:email,dept,grp,pwd_reset
```

Process supplier invoice:

```
PROC|FIN|return:FIN-Agent|p:2|aacp:1.1|res:invoice|
sup:ABC-Ltd|amt:4200|ccy:GBP|match:PO-441|terms:net30
```

Write deterministic audit record:

```
LOG|HR|return:AUD-Agent|p:2|aacp:1.1|actor:ORCHESTRATOR|
chain:HR-AGENT,FIN-AGENT,HR-AGENT,HR-AGENT|
status:review_required
```

5. Valid Field Values

5.1. TASK Values

The following TASK values are defined in AACP v1.1: FETCH, PROC, FLAG, RESOLVE, LOG, SEND, BUILD, MERGE, CALC, REPORT, ACK, SYNC.

Implementations MUST warn on unrecognised TASK values. Implementations MUST NOT reject packets with unrecognised TASK values, to permit forward compatibility.

5.2. DOM Values

The following DOM values are defined in AACP v1.1: HR, FIN, SALES, LEGAL, IT, CS, MKT.

Implementors MAY define additional domain values. Unrecognised DOM values SHOULD generate advisory warnings.

5.3. Priority Values

The p: field accepts: 1 (critical), 2 (medium, default), 3 (low).

6. Encoding

6.1. Rule-Based Encoding

For known, repetitive workflow types, a rule-based encoder deterministically produces AACP packets from structured input without an LLM API call. This approach has zero API cost and produces identical output for identical input on every run.

Reference implementations are provided for the following workflow types in the open-source SDK [AACP-SDK]:

* Payroll (PayrollEncoder, 6 coordination hops)
* IT Provisioning (ITEncoder, 6 coordination hops)
* Invoice Processing (InvoiceEncoder, 3 coordination hops)
* Contract Review (ContractEncoder, 3 coordination hops)

6.2. LLM-Assisted Encoding

For novel instructions outside known workflow patterns, an LLM-assisted encoder produces AACP packets by submitting the English instruction to a language model with the AACP specification as a system prompt. At current pricing for Claude Sonnet 4.5 this costs approximately $0.0006 USD per instruction.

6.3. Three-Tier Fallback and Registry

The FallbackEncoder routes encoding requests through three tiers in priority order:

Tier 1 -- Hash cache lookup: The SHA-256 hash of the normalised English instruction is looked up in the local registry. An exact match returns the cached packet immediately at zero cost. The times_seen counter is incremented.

Tier 2 -- Pattern match: The instruction is matched against keyword patterns from both the registry and a built-in pattern set. A match above the configured threshold returns the cached packet or routes to rule-based encoding, both at zero cost.

Tier 3 -- LLM fallback: Novel instructions with no match are encoded via LLM. The result is logged to the registry with extracted keywords, enabling Tier 1 and Tier 2 routing on all subsequent occurrences.

This architecture ensures that each novel instruction pattern incurs at most one LLM encoding call regardless of how many times the instruction subsequently appears in the system. The "one LLM call per pattern, reused indefinitely" property has been validated in the reference implementation [AACP-SDK].

7. Validation

AACP validators MUST check:

* Field 0 (TASK) is present and recognised
* Field 1 (DOM) is present and recognised
* A return: named field is present and non-empty
* An aacp: named field is present

AACP validators SHOULD warn on:

* Unrecognised TASK or DOM values
* Missing p: field
* AACP version mismatch
* sentiment: present without tone:

* ltv: present without ccy:

* Unknown extended field keys

Validation errors MUST be reported before transmission. Warnings SHOULD be logged but MUST NOT prevent transmission.

8. Decoding

AACP packets are designed to be human-readable as written. Decoders MAY expand packets into structured English for audit purposes or debugging. For audit purposes, the AACP packet itself is the canonical record. Decoded output is advisory.

9. Compression Boundaries

Coordination tokens compress well: routing instructions, resource references, structured intent, action verbs, and metadata are efficiently expressed in AACP.

Task tokens do not compress: the actual work an agent performs is unchanged by AACP. Total workflow cost impact depends on the ratio of coordination to task tokens.

Embedding field lists and URI data pointers in packets increases token count significantly and SHOULD be avoided where the receiving agent has default field definitions in its system configuration.

10. Tokenisation Benchmark

10.1. Methodology

Coordination token counts were measured using live API usage_metadata. Each message was submitted as a bare user message with no system prompt and max_tokens=1 to isolate coordination token counts. The English baseline is the full verbose instruction the AACP packet replaces in production. Benchmark date: May 2026.

10.2. Tokenisation Results

Four-hop payroll workflow. Tokens from live API.

Hop	English	AACP	Claude%	GPT-4o%
fetch employees	56	52	-7.1%	-12.7%
fetch budgets	57	47	-17.5%	-16.0%
merge and calculate	65	43	-33.8%	-31.6%
generate report	62	43	-30.6%	-33.3%
TOTAL	240	185	-22.9%	-23.7%

10.3. Multi-Model Lab Validation

A working multi-agent payroll lab [AACP-LAB] validates AACP packet interpretation across four models on a five-hop Q1 FY2026 payroll workflow. The workflow reads from real CSV data files and produces formatted Excel output via agent coordination using AACP packets throughout.

Model	Cost	Tokens in	Latency	Success
gpt-4.1-mini	$0.0044	7,566	82.5s	Y
gpt-4.1	$0.0224	7,673	39.4s	Y
claude-sonnet-4-5	$0.0416	9,062	77.5s	Y
gpt-4o	$0.0552	7,615	31.6s	Y

All four models correctly interpreted AACP v1.1 packets across all five workflow hops without model-specific prompt tuning. Every packet validated against the v1.1 schema. The deterministic audit agent (Tier 0 -- no LLM call, $0.00) wrote structured audit records on every run.

A notable quality difference was observed: Claude Sonnet 4.5 identified anomalies across all four cost centres (8 total) while GPT models identified anomalies in two. For compliance-critical workflows, model selection should consider reasoning depth alongside cost. For coordination-heavy operational workflows where task complexity is low and structured output is the primary requirement, GPT-4.1-mini at $0.0044 per five-hop run provides adequate quality at significantly lower cost.

10.4. Encoding Cost Comparison

For workflows handled by rule-based encoders, the total encoding cost per workflow run is $0.00 regardless of the number of hops. The LLM fallback encoder costs approximately $0.0006 per novel instruction. Once logged to the registry, the same instruction is subsequently served from cache at $0.00.

For comparison, encoding the same four coordination messages via direct LLM call (without AACP rule-based encoding) costs approximately $0.0024 at Claude Sonnet 4.5 pricing. Over 500 monthly payroll runs, the rule-based encoder saves approximately $1.20 in encoding costs alone, in addition to the coordination token reduction on every hop.

11. Relationship to Existing Protocols

MCP [MCP] defines how agents access external tools and resources. AACP operates inside MCP message payloads, compressing the coordination instructions that describe what to retrieve or action. The two protocols are complementary.

A2A [A2A] defines agent discovery and task routing between agents. AACP compresses the content of messages that A2A routes. The two protocols are complementary.

AACP fills a distinct layer: deterministic, typed encoding of coordination message content. No existing published protocol addresses this specific layer.

12. Implementation

A reference implementation of AACP v1.1 is available as an open-source Python package [AACP-SDK]:

```
pip install aacp
```

The package includes the rule-based encoder with four workflow encoder implementations, the three-tier fallback encoder with registry, the schema validator, and a packet decoder. The package is available at https://pypi.org/project/aacp/ and the source is published at https://github.com/MackayAndrew/aacp under the MIT licence.

A working multi-agent lab demonstrating AACP coordination across four LLM models is available at https://github.com/MackayAndrew/aacp-lab [AACP-LAB].

13. Extensibility

Unknown named fields generate advisory warnings, not errors. Implementors MAY define organisation-private fields using a namespacing convention to avoid collision with future AACP fields (e.g. org_fieldname:).

Implementors MAY define domain values beyond the seven defined in Section 5.2. A community encoding registry is planned for a future version, enabling shared rule-based patterns across implementors.

14. Version Policy

The aacp: field MUST be included in every packet and MUST specify the protocol version. Breaking changes increment the major version. Additive changes increment the minor version. Implementations SHOULD warn on version mismatch but MUST NOT reject packets on version mismatch alone.

15. Security Considerations

Packet injection: AACP packets MUST be treated as untrusted input. Validators MUST be applied before processing. Implementations MUST NOT execute actions based on unvalidated packets.

Prompt injection: malicious content in AACP field values could be interpreted as instructions by a receiving LLM agent. Implementations SHOULD sanitise field values before including them in LLM prompts. Field values SHOULD be treated as data, not instructions.

Replay attacks: AACP v1.1 does not include sequence numbers or timestamps. Security-sensitive environments SHOULD add replay protection at the transport layer.

Sensitive data: PII, financial records, and credentials MUST NOT be embedded in AACP field values. Use authenticated data access layers referenced by pointer instead.

Registry security: the local registry stores previously encoded instructions and their resulting packets. The registry file SHOULD be access-controlled and SHOULD NOT be committed to version control when it contains sensitive workflow instructions.

16. IANA Considerations

This document has no IANA actions.

17. References

17.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

17.2. Informative References

[ECOLANG] Mou, Y., "EcoLANG: Towards Efficient Agent Communication via Evolved Language", arXiv:2505.06904, May 2025, <https://arxiv.org/abs/2505.06904>.

[MCP] Anthropic, "Model Context Protocol", 2024, <https://modelcontextprotocol.io/>.

[A2A] Google, "Agent-to-Agent Protocol", 2025, <https://google.github.io/A2A/>.

[AACP-SDK] Mackay, A., "AACP Python SDK", Available at https://pypi.org/project/aacp/, 2026, <https://github.com/MackayAndrew/aacp>.

[AACP-LAB] Mackay, A., "AACP Multi-Agent Lab", Four-model payroll workflow validation. Q1 FY2026 payroll close with Excel output., 2026, <https://github.com/MackayAndrew/aacp-lab>.

Author's Address

Andrew Mackay
Independent
Email: mackayandrewr@gmail.com
URI: https://aacp.dev