Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 9 October 2026

RB. Benzing
April 2026

Agent Context Compression Protocol (ACCP)
draft-benzing-accp-00

Abstract

This document specifies the Agent Context Compression Protocol (ACCP), a semantic encoding and context management protocol for communication between AI agents within agentic harnesses. ACCP defines a compact message format, intent ontology, state compression strategy, and codec interface that collectively reduce token consumption by 60-90% compared to natural language or standard JSON messaging. ACCP is designed to complement existing protocols (MCP, A2A) and is transport-agnostic.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 3 October 2026.

Copyright Notice

Table of Contents

1. Introduction 3
1.1. Motivation 3
1.2. Scope 3
1.3. Terminology 4
2. Protocol Overview 4
2.1. Architecture 4
2.2. Frame Lifecycle 5
3. Message Format (ACCP-M) 5
3.1. Frame Structure 5
3.1.1. Typed Value Canonical Rules 6
3.2. Frame Examples 7
3.3. Token Optimization Rules 7
3.4. Frame Size Limits 8
3.5. Standard Message Envelope 8
3.6. Error Model 9
3.7. Delivery Semantics 11
4. Intent and Operation Ontology (ACCP-I) 12
4.1. Core Intents 12
4.2. Standard Key Abbreviations 13
4.3. Operation Registry 14
5. Context State Management (ACCP-S) 14
5.1. State Hierarchy 14
5.2. Compression Checkpoints 14
5.3. Checkpoint Process 15
5.4. Delta Encoding 15
6. Schema Registry (ACCP-R) 15
6.1. Purpose 15
6.2. Registry Structure 15
6.3. Usage in Frames 16
6.4. Registry Negotiation 16
7. Codec API Specification 16
7.1. Interface Definition 16
7.2. Encoding Pipeline 17
7.3. Decoding Pipeline 17
8. Conformance Requirements 17
8.1. Minimum Conformance (Level 1) 17
8.2. Standard Conformance (Level 2) 18

8.3. Full Conformance (Level 3) 18
9. Security Considerations 18
10. Domain Profiles 19
10.1. Chat Profile (CH) 19
10.2. Tool / MCP Profile (TC) 19
10.3. Transaction Profile (TX) 19
10.4. Streaming Profile (ST) 20
10.5. Workflow Profile (TA) 20
11. Transport Bindings 20
11.1. ACCP-over-HTTP 20
11.2. ACCP-over-WebSocket 21
11.3. ACCP-over-MCP 21
11.4. ACCP-over-A2A 21
12. IANA Considerations 21
12.1. Media Type Registration 21
13. Normative References 22
14. Informative References 22
Appendix A. Token Comparison Benchmarks 22
Appendix B. Future Work 23
Acknowledgements 23
Author’s Address 23

1. Introduction

1.1. Motivation

The operational cost of multi-agent AI systems is dominated by token consumption at the LLM inference layer. Existing agent communication protocols optimize for interoperability, discovery, and routing but transmit messages in verbose formats (natural language, pretty-printed JSON) that are highly wasteful from a token-cost perspective.

ACCP addresses this by introducing a protocol layer between the orchestration harness and the LLM API that encodes agent communication into a compact, semantically-faithful format.

1.2. Scope

ACCP defines:

* Message encoding format (Section 3)

* Intent and operation codes (Section 4)

* Context state management (Section 5)

* Schema registry interface (Section 6)

* Codec API (Section 7)

ACCP does NOT define:

* Transport mechanisms (defer to [MCP]/[A2A]/HTTP)

* Agent discovery or routing (defer to [A2A]/ANP)

* LLM API specifics

* Authentication or authorization (defer to transport layer)

1.3. 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 capitalized, as shown here.

Term	Definition
Agent	An autonomous LLM-backed process within a harness
Harness	The orchestration framework managing agents
Codec	The ACCP encoder/decoder library
Frame	A single ACCP-encoded message unit
Session	A bounded sequence of frames comprising one workflow
Checkpoint	A state compression boundary
Intent	A standardized operation code

Table 1

2. Protocol Overview

2.1. Architecture

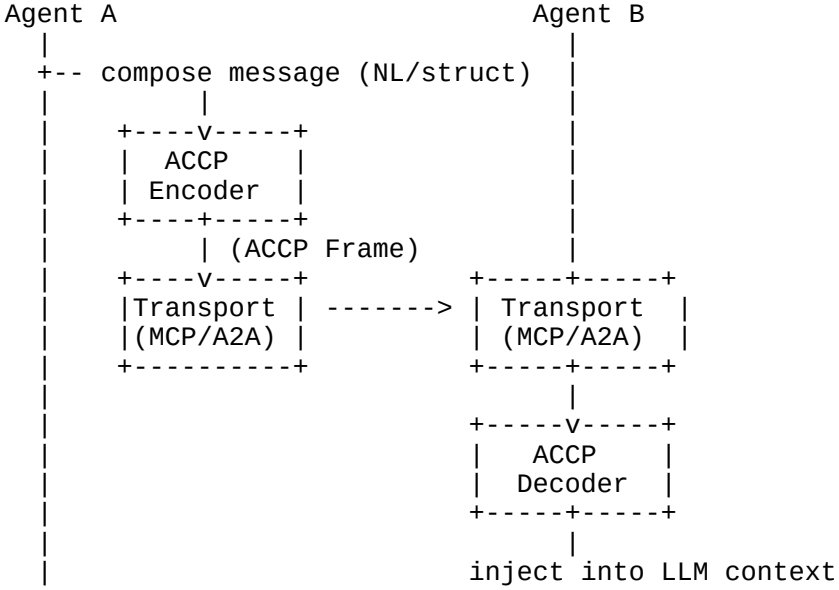

Figure 1: ACCP Architecture

2.2. Frame Lifecycle

1. Sending agent composes a message (natural language or structured)
2. ACCP Encoder compresses it into a Frame
3. Frame is transmitted via the existing transport layer
4. ACCP Decoder at receiving end reconstructs the semantic content
5. Decoded content is injected into the receiving agent’s LLM context

3. Message Format (ACCP-M)

3.1. Frame Structure

An ACCP Frame is a single-line UTF-8 string defined by the following ABNF grammar ([RFC5234]):

```
frame       = header body [metadata]
header      = "@" agent-id ">" intent
body        = ":" operation "{" [payload] "}"
metadata    = "[" meta-pairs "]"
payload     = param *( "|" param )
param       = key ":" value
value       = typed-literal / array / map / ref / null
typed-literal = boolean / integer / decimal / string
boolean     = "true" / "false"
integer     = ["-"] 1*DIGIT
decimal     = ["-"] 1*DIGIT "." 1*DIGIT
string      = 1*( safe-char / escaped-char )
safe-char   = VCHAR except delimiter
escaped-char = "\" delimiter
delimiter   = "@" / ">" / ":" / "{" / "}" / "[" / "]"
            / "|" / "$" / "," / "~" / "\"
null        = "~"
array       = "[" [ value *( "," value ) ] "]"
map         = "{" [ key ":" value *( "," key ":" value ) ] "}"
ref         = "$" ref-key
meta-pairs  = param *( "," param )

agent-id    = 1*( ALPHA / DIGIT / "-" / "_" )
intent      = 1*( ALPHA )
operation   = 1*( ALPHA / DIGIT / "_" )
key         = 1*( ALPHA / DIGIT / "_" )
ref-key     = 1*( ALPHA / DIGIT / "_" / "." )
```

3.1.1. Typed Value Canonical Rules

Type	Wire Format	Example
String	Raw printable ASCII; delimiters escaped with \	hello, a\:b
Integer	Decimal, no leading zeros	42, -7
Decimal	Canonical 6dp, trailing zeros stripped	3.14, -0.5
Boolean	true or false	true
Null	~	~
Array	[v1,v2,v3]	[1,2,3]
Map	{k1:v1,k2:v2} -- keys	{a:1,b:2}

	sorted ascending		
Ref	$tier.key	$warm.ckpt_1.status	

Table 2

Decimal values MUST be serialized with up to 6 decimal places, trailing zeros stripped. Encoders MUST NOT use scientific notation.

3.2. Frame Examples

Task completion with findings:

```
@research>done:analyze{d:q3_sales|f:[rev:-12%QoQ,ent_seg:decline,
  churn:+3.2%]|nx:@strategy:plan}
```

Request with parameters:

```
@planner>req:schedule{who:@dev_team|when:sprint_14|
  task:impl_auth_module|pri:high}
```

Query with context reference:

```
@analyst>qry:lookup{src:$ctx.sales_db|q:revenue_by_region|fmt:summary}
```

State sync:

```
@orchestrator>sync:state{v:7|delta:{task_3:done,task_4:wip,
  budget:$42.30}}
```

Error with escalation:

```
@data_agent>fail:fetch{src:api.crm|err:timeout_30s|retry:3|
  esc:@supervisor}
```

3.3. Token Optimization Rules

Encoders MUST follow these rules to minimize token count:

1. No whitespace -- Frames MUST NOT contain spaces, tabs, or newlines.

2. Abbreviated keys -- Use shortest unambiguous key names (see Section 4.2).

3. Symbolic values -- Prefer symbols over words: +, -, %, >, <.

4. Reference over inline -- If a value exceeds 50 characters, store in state and reference via $key.

5. Omit defaults -- Do not encode values that match the schema default.

6. Coalesce arrays -- Use comma separation within brackets, no spaces.

3.4. Frame Size Limits

Component	Max Tokens (soft)	Max Tokens (hard)
Header	5	10
Body	50	200
Metadata	10	30
Total Frame	65	240

Table 3

Frames exceeding the hard limit SHOULD be split into a frame sequence or reference external state.

3.5. Standard Message Envelope

Every ACCP frame MUST include the following envelope fields in its metadata block:

Field	Abbrev	Type	Required	Description
msg_id	mid	string(12)	MUST	Unique message identifier (hex, 12 chars)
sequence	seq	integer	MUST	Monotonically increasing per session
timestamp	ts	integer	MUST	Unix epoch seconds
correlation_id	cid	string	SHOULD	Links request to response
causation_id	aid	string	MAY	Links to the message that caused this one
session_id	sid	string	SHOULD	ACCP session identifier
ttl	ttl	integer	MAY	Seconds until frame expires; 0 = no expiry

Table 4

Example metadata block:

[mid:49679033e07c,seq:3,ts:1714000000,cid:corr123,sid:abc-session]

Decoders MUST reject frames with expired TTL. Frames missing "mid" or "seq" MUST be rejected.

3.6. Error Model

Standard error frames use intent "fail", operation "error", and schema "ER".

Error code taxonomy:

Range	Category
E1xxx	Parse / grammar errors
E2xxx	State store errors
E3xxx	Transport / delivery errors
E4xxx	Tool / MCP errors
E5xxx	Policy / authorization errors
E9xxx	Internal / unknown errors

Table 5

Standard error codes:

Code	Name	Retryable
E1001	PARSE_ERROR	No
E1002	INVALID_INTENT	No
E1003	UNKNOWN_SCHEMA	No
E1004	INVALID_TYPE	No
E2001	REF_NOT_FOUND	No
E2002	REF_EXPIRED	No
E2003	BUDGET_EXCEEDED	No
E3001	TIMEOUT	Yes
E3002	DUPLICATE	No
E3003	SEQUENCE_GAP	Yes
E4001	TOOL_NOT_FOUND	No
E4002	TOOL_EXEC_FAILED	Yes
E4003	TOOL_SCHEMA_MISMATCH	No
E5001	POLICY_DENIED	No
E5002	UNAUTHORIZED_REF	No
E9999	INTERNAL_ERROR	Yes

Table 6

Error frame example:

```
@agent>fail:error{code:E3001|msg:connection_timed_out|retry:true|
  schema:ER}[mid:abc,seq:4,ts:1714000001]
```

3.7. Delivery Semantics

Implementations MUST enforce the following delivery guarantees:

1. Idempotency: Decoders MUST maintain a set of seen "msg_id" values per session. Frames with a previously seen "msg_id" MUST be rejected with error E3002 DUPLICATE.

2. Ordering: Frames MUST be processed in ascending "seq" order. A gap in sequence numbers MUST trigger error E3003 SEQUENCE_GAP.

3. Retries: Frames with retryable error codes MAY be retransmitted with the same "correlation_id" and a new "msg_id" and "seq".

4. Cancellation: A "cancel" intent frame referencing a "correlation_id" MUST stop all processing for that request chain.

5. TTL enforcement: Frames received after their TTL has elapsed MUST be rejected silently (no error response, to prevent timing attacks).

4. Intent and Operation Ontology (ACCP-I)

4.1. Core Intents

Code	Meaning	Description
req	Request	Request another agent to perform an action
done	Complete	Report task completion with results
fail	Failure	Report task failure with error details
wait	Waiting	Awaiting external input or dependency
esc	Escalate	Delegate to a higher-authority agent
comp	Compress	Trigger context compression checkpoint
sync	Synchronize	Share or request state synchronization
qry	Query	Request information lookup

ack	Acknowledge	Confirm receipt of a frame
cancel	Cancel	Abort an in-progress task
stream	Stream	Begin streaming incremental results
end	End stream	Close a streaming sequence

Table 7

4.2. Standard Key Abbreviations

Abbreviation	Full Meaning
d	data / dataset
f	findings / fields
nx	next action
src	source
dst	destination
q	query
fmt	format
pri	priority
err	error
v	version
ts	timestamp
ttl	time-to-live
ctx	context reference
who	target agent/entity

when	temporal constraint
why	rationale code

Table 8

4.3. Operation Registry

Operations are domain-specific verbs registered in the Schema Registry (Section 6). Core operations:

analyze, plan, execute, review, approve, reject, fetch, transform, summarize, classify, generate, validate, schedule, notify, merge, split, route, cache, purge

Harnesses MAY register custom operations via the registry extension mechanism.

5. Context State Management (ACCP-S)

5.1. State Hierarchy

ACCP defines a three-tier state model:

Tier 1: HOT STATE (in-context) Current frame + immediate parent frames. Active task parameters. Max budget: 500 tokens.

Tier 2: WARM STATE (compressed summaries) Checkpoint summaries from completed phases. Key-fact extractions. Referenced via $warm.key. Max budget: 200 tokens per checkpoint.

Tier 3: COLD STATE (external store) Full conversation history. Raw tool outputs. Referenced via $cold.key. Zero token budget (not injected into context).

5.2. Compression Checkpoints

A checkpoint MUST be triggered when ANY of the following conditions are met:

1. An agent completes a task ("done" intent)

2. Accumulated hot state exceeds 400 tokens

3. A handoff between agents occurs ("esc" intent)

4. The harness signals a phase boundary

5. An agent explicitly requests compression ("comp" intent)

5.3. Checkpoint Process

1. Freeze current hot state
2. Extract key facts (entity, value, relationship triples)
3. Generate compressed summary (less than or equal to 100 tokens)
4. Move summary to warm state with checkpoint ID
5. Move raw data to cold state
6. Reset hot state with checkpoint reference

5.4. Delta Encoding

After checkpoint N, subsequent frames SHOULD transmit only deltas:

```
@agent>sync:state{v:N+1|delta:{changed_key:new_val,removed_key:null}}
```

The decoder reconstructs full state by applying deltas to the last checkpoint.

6. Schema Registry (ACCP-R)

6.1. Purpose

The schema registry allows agents to reference pre-defined data structures by compact codes rather than re-describing them in every message. This eliminates the need to re-inject tool definitions and response schemas into the context window.

6.2. Registry Structure

```
{
  "schemas": {
    "sales_report": {
      "code": "SR",
      "version": 1,
      "fields": ["period", "revenue", "growth_pct",
                 "segments", "notes"],
      "defaults": { "period": "quarterly", "segments": [] }
    },
    "task_assignment": {
      "code": "TA",
      "version": 2,
      "fields": ["assignee", "task", "priority",
                  "deadline", "deps"],
      "defaults": { "priority": "medium", "deps": [] }
    }
  }
}
```

6.3. Usage in Frames

```
@planner>req:execute{schema:TA|assignee:@dev|task:auth_module|
  deadline:sprint_14}
```

The decoder expands "schema:TA" into the full field set, populating defaults for omitted fields.

6.4. Registry Negotiation

At session start, agents MUST exchange registry versions:

```
@orchestrator>sync:registry{v:3|hash:a7f2c1}
```

If hashes match, no further exchange is needed. On mismatch, the full registry delta is transmitted once and cached.

7. Codec API Specification

7.1. Interface Definition

The codec interface defines the following operations that conforming implementations MUST provide:

Encode(message, session) -> Frame Encode a structured message into an ACCP Frame

Decode(frame, session) -> Message Decode an ACCP Frame back into a structured message

Checkpoint(session) -> CheckpointResult Trigger a compression checkpoint

GetBudget(session) -> TokenBudget Get current token budget usage

RegisterSchema(name, schema) Register a custom schema

7.2. Encoding Pipeline

1. Schema resolution (expand or compact via registry)
2. Key abbreviation (full keys to abbreviated keys)
3. Value compression (inline values to references if large)
4. Default omission (remove fields matching schema defaults)
5. Token estimation (count via BPE tokenizer)
6. Budget check (within frame limits): if YES, emit single frame; if NO, split into frame sequence or externalize to state
7. Emit ACCP Frame string

7.3. Decoding Pipeline

1. Parse grammar (header, body, metadata)
2. Resolve references ($warm.key, $cold.key, schema codes)
3. Expand abbreviations (abbreviated keys to full keys)
4. Apply schema defaults (fill omitted fields)
5. Reconstruct structured message
6. Return structured message for LLM context injection

8. Conformance Requirements

8.1. Minimum Conformance (Level 1)

An implementation MUST support:

* Frame encoding/decoding (Section 3)
* Core intents (Section 4.1)

* Basic key abbreviations (Section 4.2)

8.2. Standard Conformance (Level 2)

An implementation MUST additionally support:

* Context state management (Section 5)
* Compression checkpoints (Section 5.2)
* Schema registry (Section 6)

8.3. Full Conformance (Level 3)

An implementation MUST additionally support:

* Delta encoding (Section 5.4)
* Registry negotiation (Section 6.4)
* Token budget enforcement (Section 7.1)
* Streaming frames (Section 4.1, "stream"/"end" intents)

9. Security Considerations

1. Opacity risk: Compact formats are harder for humans to audit. Implementations SHOULD provide a "debug mode" that logs decoded human-readable equivalents.
2. Injection: Frame parsing MUST validate grammar strictly. Malformed frames MUST be rejected, not partially parsed.
3. State store access: Cold state references MUST be scoped to the current session. Cross-session state access requires explicit authorization.
4. Registry tampering: Schema registries SHOULD be integrity-checked via hash (Section 6.4).
5. Parser DoS Protection: Decoders MUST implement strict limits on array nesting depth (recommended maximum: 5 levels) and limit payload parsing complexity to prevent resource exhaustion attacks.

10. Domain Profiles

A Domain Profile is a named combination of schema codes, intent sequences, and operational patterns optimised for a specific use case. Profiles are additive -- a harness may activate multiple profiles simultaneously.

10.1. Chat Profile (CH)

Schema: CH -- fields: role, content, turn, lang, reply_to. Defaults: role=assistant, lang=en.

```
@user>req:chat{content:What_are_Q3_findings?|role:user|turn:1|
  schema:CH}[mid:...,seq:1]
@assistant>done:chat{content:Revenue_declined_12%.|turn:2|
  schema:CH}[mid:...,seq:2,cid:...]
```

10.2. Tool / MCP Profile (TC)

Schema: TC -- fields: tool_name, arguments, result, status, error_code. Defaults: status=ok.

```
@orchestrator>req:tool{tool:web_search|args:{q:ACCP,max:5}|
  schema:TC}[mid:m1,seq:1]
@tool_agent>done:tool{tool:web_search|res:{hits:[...]}|stat:ok|
  schema:TC}[mid:m2,seq:2,cid:m1]
```

Tool calls MUST include "correlation_id" in the response to link result to request. Tool errors MUST use schema "ER" with code E4002 TOOL_EXEC_FAILED.

10.3. Transaction Profile (TX)

Schema: TX -- fields: transaction_id, amount, currency, account, reference, status, retryable. Defaults: currency=USD, status=pending, retryable=false.

```
@payments>req:transaction{txn:txn_001|amt:142.5|acc:acct_9876|
  schema:TX}[mid:...,seq:5]
@payments>done:transaction{txn:txn_001|stat:settled|
  schema:TX}[mid:...,seq:6,cid:...]
```

Transaction safety requirements:

* "transaction_id" MUST be stable across retries (idempotency key).

* "amount" MUST be serialized as decimal (not string) to avoid precision loss.

* "currency" SHOULD follow ISO 4217.

* Retries MUST use the same "transaction_id" with a new "msg_id".

10.4. Streaming Profile (ST)

Schema: ST -- fields: chunk_index, total_chunks, data, is_final.
Defaults: is_final=false.

```
@streamer>stream:infer{idx:0|tot:3|d:Hello|
  schema:ST}[mid:m1,seq:1,cid:stream_abc]
@streamer>stream:infer{idx:1|tot:3|d:_world|
  schema:ST}[mid:m2,seq:2,cid:stream_abc]
@streamer>stream:infer{idx:2|tot:3|d:!|done:true|
  schema:ST}[mid:m3,seq:3,cid:stream_abc]
```

All chunks in a stream MUST share the same "correlation_id". The final chunk MUST set "is_final=true". Consumers MUST buffer and reorder by "chunk_index" before processing.

10.5. Workflow Profile (TA)

Schema: TA -- fields: assignee, task, priority, deadline, deps.
Defaults: priority=medium, deps=[].

```
@planner>req:schedule{asgn:@dev|task:impl_auth|dead:sprint_14|
  pri:high|schema:TA}[mid:...,seq:8]
@dev>done:schedule{task:impl_auth|stat:complete|prog:100|
  schema:TA}[mid:...,seq:9,cid:...]
```

11. Transport Bindings

ACCP frames are transport-agnostic. The following bindings define how frames are carried over common transports.

11.1. ACCP-over-HTTP

* Method: POST /accp/v1/frames

* Content-Type: application/accp

* Body: One ACCP frame string per request (UTF-8)

* Response: 200 OK with response ACCP frame, or 400 with ER-schema error frame body

* Streaming: use Transfer-Encoding: chunked with one ST-schema frame per chunk

11.2. ACCP-over-WebSocket

* Frame type: text
* One ACCP frame string per WebSocket message
* Session established by the first sync:registry frame from the initiator
* Heartbeat: ping/pong intent frames every 30 seconds

11.3. ACCP-over-MCP

* ACCP frames are carried as MCP resource content with MIME type application/accp
* Tool calls use the TC profile; results are returned as MCP tool responses containing the TC-schema done:tool frame
* Registry sync occurs at MCP session initialization via a sync:registry frame in the system prompt

11.4. ACCP-over-A2A

* ACCP frames are embedded as A2A message parts with content type application/accp
* A2A task creation maps to ACCP "req" intent; task completion maps to "done"
* A2A streaming artifact parts map 1:1 to ST-schema stream frames

12. IANA Considerations

This document requests IANA registration of the media type "application/accp" for ACCP frame payloads.

12.1. Media Type Registration

Type name: application

Subtype name: accp

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: UTF-8

Security considerations: See Section 9 of this document

Interoperability considerations: N/A

Published specification: This document

13. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, 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, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

14. Informative References

[MCP] Anthropic, "Model Context Protocol", 2024.

[A2A] Google, "Agent2Agent (A2A) Protocol Specification", 2025.

[ACP] IBM, "Agent Communication Protocol (ACP)", 2025.

Appendix A. Token Comparison Benchmarks

Scenario	NL Tokens	JSON Tokens	ACCP Tokens	Reduction
Task completion + findings	85	62	22	74%
Task request + params	45	38	15	67%
Error + escalation	60	48	18	70%
State sync (10 fields)	120	95	35	71%

Full 5-agent pipeline	8,500	6,200	1,800	79%

Table 9

Note: Benchmarks are based on empirical validation of Phase 1 implementation.

Appendix B. Future Work

* Model-specific tokenizer profiles: Optimize encoding per model's BPE vocabulary.

* Adaptive compression: ML-based encoder that learns optimal compression per domain.

Acknowledgements

The author acknowledges the contributions of the broader AI agent protocol community, including the developers of [MCP], [A2A], and [ACP], whose work informed the design of ACCP.

Author's Address

Russell Benzing
Email: me@russellbenzing.com