AXTP: Agent Experience Transfer Protocol

A Protocol for Structured Experience Capture, Validation, and Transfer Between Autonomous AI Agents

Richard Edwards

The Abstraction Stack

February 2026

Version 0.1 — Working Draft

ABSTRACT

As AI agents are deployed at scale across enterprise and consumer environments, a critical infrastructure gap has emerged: there is no standardized mechanism for agents to capture, validate, and share structured execution experience. Each agent execution remains isolated, with experiential knowledge discarded upon session termination. This paper introduces the Agent Experience Transfer Protocol (AXTP), an open protocol that defines structured Experience Records (XRs) for encoding execution knowledge, managed Experience Pools for shared storage and retrieval, and a trust and governance framework for validating experience accuracy, detecting adversarial manipulation, and maintaining auditability. AXTP operates as a complementary layer in the emerging agent protocol stack, sitting above existing protocols including MCP, A2A, and ACP. Initial reference implementation benchmarks demonstrate a 65% reduction in task completion time when agents access pooled prior experience, suggesting significant efficiency gains from structured experiential knowledge transfer. The protocol specification, governance framework, and reference implementation are published as open source to enable broad adoption and community scrutiny.

Keywords: AI agents, experience transfer, agent memory, compound intelligence, agent governance, trust scoring, multi-agent systems, agent protocols

TABLE OF CONTENTS

1. Introduction

1.1 The Agent Amnesia Problem

The deployment of autonomous AI agents has accelerated dramatically. Organizations now rely on agents to execute complex tasks including code generation, API integrations, data analysis, customer service, workflow automation, and multi-step reasoning chains. These agents operate across diverse frameworks — LangChain, AutoGen, CrewAI, custom implementations — built on foundation models from Anthropic, OpenAI, Google, and others.

Despite this proliferation, a fundamental architectural limitation persists: **agent execution experience is transient**. When an agent completes a task — whether successfully or through failure and recovery — the experiential knowledge generated during that execution is discarded. Each subsequent invocation begins from a zero-knowledge state with respect to prior executions by the same agent or any other agent in the network.

This is structurally analogous to an organization where every employee starts their first day, every day. No institutional knowledge transfers. No lessons learned from prior projects accumulate. No onboarding materials exist. The cost of this design choice compounds with scale: as organizations deploy more agents across more tasks, the aggregate waste from repeated errors, redundant exploration, and unshared discoveries grows proportionally.

1.2 Limitations of Current Approaches

Three approaches currently attempt to address agent knowledge persistence, each with significant limitations:

Stateless Execution: The default mode for most agent frameworks. Each invocation is independent, with no persistent state between runs. This is simple and safe but ensures that agents are perpetually junior — unable to build on prior experience. An agent that successfully navigated a complex Stripe API integration yesterday provides zero benefit to the agent assigned the same integration today.

Individual Agent Memory: Systems such as ChatGPT's memory and Claude's memory features maintain per-agent conversation history. These systems are valuable for personalization but fundamentally limited: they create knowledge silos where Agent A's hard-won insights never reach Agent B. They lack governance controls for accuracy verification. They conflate conversational context with transferable operational knowledge.

Model Fine-tuning: Knowledge can be incorporated into an agent's underlying model through fine-tuning or reinforcement learning from human feedback (RLHF). This approach is computationally expensive, slow to implement, opaque in its effects on model behavior, difficult to audit, and impossible to selectively revert. Fine-tuning addresses knowledge accumulation at the model level but cannot provide the task-specific, auditable, real-time knowledge transfer that operational agent networks require.

None of these approaches provide a mechanism for **structured, validated, cross-agent experiential knowledge transfer with governance controls**. This paper argues that such a mechanism is the primary barrier to compound intelligence in multi-agent systems, and proposes a protocol to address it.

1.3 Contribution

This paper makes the following contributions:

(1) We identify and characterize the experience transfer gap in the current agent protocol stack, demonstrating that existing protocols address tool connectivity, task delegation, and message structure but not experiential knowledge transfer.

(2) We propose AXTP, an open protocol defining structured Experience Records (XRs), managed Experience Pools, and four core operations (Deposit, Retrieve, Validate, Inspect) for cross-agent experience exchange.

(3) We present a trust and governance framework — incorporating composite trust scoring, poison detection, amplification attack mitigation, and audit trails — designed to make shared agent experience safe and reliable at scale.

(4) We provide an open-source reference implementation with initial demonstration results showing 65% task completion time reduction when agents access pooled prior experience.

2. Related Work

2.1 Agent Protocol Landscape

The agent protocol ecosystem is developing rapidly, with several standards emerging to address different layers of the agent communication stack:

Protocol	Maintainer	Function	Experience Transfer
MCP	Anthropic	Tool & context connection	No
A2A	Google	Agent-to-agent delegation	No
ACP	IBM	Structured agent communication	No
Agent Protocol	AI Eng. Foundation	Universal agent API	No
AG-UI	CopilotKit	Agent-user interaction	No
ATXP	ATXP Dev	Agent transaction payments	No

AXTP (this work)	**Edwards**	**Experience transfer & compounding**	**Yes**

Table 1: Agent protocol landscape. AXTP is the first protocol to address structured experience transfer.

Anthropic's Model Context Protocol (MCP) standardizes how agents connect to external tools, databases, and data sources, establishing a client-server model for context passing. Google's Agent-to-Agent Protocol (A2A) enables agents to discover, delegate to, and coordinate with other agents. IBM's Agent Communication Protocol (ACP) structures the message payloads exchanged between agents. Each of these protocols addresses a necessary layer of the agent stack, but none provides a mechanism for capturing or transferring execution experience.

2.2 Agent Memory Systems

Several approaches to agent memory exist in current systems. Conversational memory systems (e.g., ChatGPT memory, Claude memory) persist user preferences and interaction history within a single agent-user context. Retrieval-Augmented Generation (RAG) systems augment agent capabilities with external knowledge stores, but these are typically static document repositories rather than dynamic experiential knowledge. Vector databases such as Pinecone, Weaviate, and Chroma provide semantic search over embedded content but lack the structured governance, trust, and validation layers necessary for reliable cross-agent experience transfer.

The MemGPT system [Packer et al., 2023] introduces a virtual context management approach that extends effective context length, but operates within a single agent's context rather than across agent networks. Reflexion [Shinn et al., 2023] enables agents to learn from verbal self-reflection on prior failures, but this learning remains internal to individual agent instances.

2.3 Positioning of AXTP

AXTP is designed to complement rather than compete with existing protocols. It operates at a distinct layer in the protocol stack:

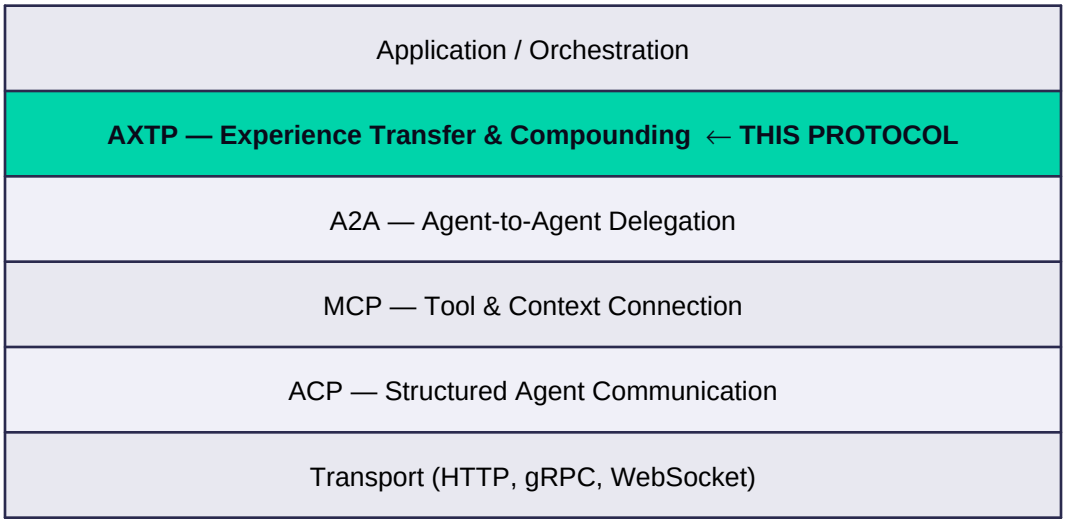

Figure 1: AXTP in the agent protocol stack. Agents use MCP to connect, A2A to delegate, and AXTP to learn.

3. Protocol Architecture

AXTP defines three core constructs — Experience Records, Experience Pools, and Protocol Operations — along with design principles that guide the protocol's architecture.

Design Principles: AXTP is agent-agnostic (works with any framework or LLM provider), transport-agnostic (defines data structures and interfaces, not transport mechanisms), governance-first (trust and auditability are core, not bolt-on), incrementally adoptable (agents can produce XRs without consuming them, and vice versa), and privacy-aware (experience is scopeable, anonymizable, and access-controlled).

3.1 Experience Records (XRs)

The Experience Record is the atomic unit of the protocol. An XR is a structured data artifact produced by an agent after task execution, designed specifically for cross-agent knowledge transfer. Unlike raw execution logs or conversational memory, an XR encodes transferable operational knowledge in a format optimized for retrieval and application by other agents.

An Experience Record comprises five components:

Context Object: Captures the task objective, execution environment (agent framework, underlying model, available tools, constraints), the trigger that initiated the task (user request, scheduler, another agent), and optional references to parent tasks in hierarchical execution chains.

Execution Trace: Records the sequence of steps taken, including for each step: the action performed, the reasoning behind it, tools used, summarized inputs and outputs (never raw data), duration, and success status. Critically, the trace records **pivots** — instances where the agent changed its approach — with explicit documentation of the reason, creating a decision audit trail.

Outcome Object: Records final status (success, partial success, failure, or aborted), a result summary, detailed error information when applicable (including recovery attempts and their outcomes), and the agent's self-assessed quality score.

Learnings Object: The highest-value component for cross-agent transfer. Encodes: **effective patterns** (strategies that worked, with applicability conditions and confidence scores), **antipatterns** (strategies that failed, with trigger conditions and recommended alternatives), **environmental notes** (context-specific observations that may not be documented elsewhere), and **recommendations** for future agents.

Trust Metadata: A composite trust score, validation status (pending, validated, or disputed), and references to validators who have assessed the record's accuracy. Trust metadata

evolves over the XR's lifetime through the governance framework described in Section 4.

3.2 Experience Pools

Experience Records are deposited into managed repositories called Experience Pools. Each pool defines a scope (global, organizational, team, or task-type specific), access policies (role-based controls for read, write, validate, and admin operations), governance configuration (validation requirements, confidence thresholds, conflict resolution strategies), and retention policies (maximum age, maximum records, archival strategy).

Pools serve as the organizational boundary for experience sharing. An enterprise might maintain separate pools for engineering tasks, customer service interactions, and data analysis workflows, each with governance policies appropriate to the domain's risk profile and rate of environmental change.

3.3 Protocol Operations

AXTP defines four core operations:

DEPOSIT(pool_id, experience_record) → deposit_receipt: An agent submits an XR to one or more pools after task execution. The operation validates the XR against the protocol schema, verifies agent authorization, executes governance checks, assigns an initial trust score based on the source agent's reputation, and returns a receipt.

RETRIEVE(pool_id, query) → ranked_experience_set: Before execution, an agent queries a pool for relevant prior experience. Query parameters include task type (with hierarchical prefix matching), context similarity, minimum confidence threshold, result limits, recency weighting, and outcome filters. Returns XRs ranked by composite relevance.

VALIDATE(pool_id, xr_id, validation) → updated_trust: Validators (agents or humans) confirm, dispute, or amend existing XRs. Validation updates trust metadata and contributes to source agent reputation. Self-validation is prohibited.

INSPECT(pool_id) → pool_metadata: Returns pool health statistics including total records, contributing agent count, average confidence, outcome distributions, and validation status distributions.

3.4 Integration with Existing Protocols

AXTP is designed for seamless integration with the existing agent protocol stack. An MCP server implementation of AXTP would expose Deposit and Retrieve as tools that any MCP-compatible agent can invoke. In an A2A context, agents can include relevant XR references when delegating tasks, enabling the receiving agent to access pre-filtered experience. AXTP can also consume events from MCP and A2A interactions to automatically generate Experience Records, reducing the burden on agent developers.

4. Trust and Governance Framework

The governance framework is the critical differentiator between AXTP and a simple shared database. Without governance, agent experience pools become attack surfaces — vulnerable to poisoning, drift, amplification, and exfiltration. This section describes the mechanisms that make shared agent experience safe and reliable.

4.1 Trust Model

Every Experience Record carries a composite trust score $T \in [0, 1]$, computed as a weighted combination of five factors:

Factor	Weight	Description
Source Reputation (R)	0.30	Historical accuracy of the contributing agent
Validation Signals (V)	0.25	Confirmations and disputes from independent validators
Outcome Correlation (O)	0.25	Performance improvement in consuming agents
Recency (D)	0.10	Temporal freshness with exponential decay
Consistency (C)	0.10	Alignment with other XRs for similar tasks

Table 2: Trust score components with default weights. Weights are configurable per pool.

The composite trust score is computed as: $T = w_R R + w_V V + w_O O + w_D D + w_C C$, where weights sum to 1.0 and are configurable per pool to reflect organizational risk tolerance. The recency factor D applies exponential decay: $D(t) = e^{-\lambda t}$, where λ is a configurable decay rate and t is time since deposit. Fast-changing domains should use higher decay rates.

Agent reputation is scoped to pools — an agent may have high reputation in one domain and low in another. New agents begin with a baseline reputation of 0.5, which adjusts based on confirmed versus disputed XRs and downstream outcome correlation.

4.2 Poison Detection

The governance layer must detect and mitigate three categories of adversarial manipulation:

Deliberate Poisoning: Malicious agents inject false experience to degrade others' performance. Mitigated by: low base reputation for new agents (limiting initial impact), statistical outlier detection on XR content, peer validation crosschecks, and rate limiting on deposits.

Drift Poisoning: Initially valid experience becomes incorrect as environments change. Mitigated by: temporal trust decay, active re-validation triggers for high-use XRs, and consumer feedback loops.

Amplification Attacks: Colluding agents validate each other's records to inflate trust scores. Mitigated by: discounting validation weight from agents with correlated deposit histories, requiring independent validation from diverse sources, and graph analysis to detect validation clusters.

4.3 Conflict Resolution

When XRs report contradictory learnings for similar tasks, pools employ one of three configurable resolution strategies: **latest-wins** (most recent XR takes precedence, suitable for fast-changing environments), **consensus** (majority of XRs determines accepted knowledge), or **administrative review** (conflicts are flagged for human resolution, appropriate for high-stakes domains).

4.4 Audit Trail

Every protocol operation — deposit, retrieve, validate, inspect — is logged in an append-only, tamper-evident audit trail. Each entry records the timestamp, actor identity, operation type, affected XR identifiers, and outcome. The audit trail supports compliance with AI governance frameworks including the EU AI Act and the NIST AI Risk Management Framework (AI RMF), and enables forensic analysis of experience pool integrity.

5. Security Analysis

5.1 Threat Model

The AXTP threat model addresses six primary attack vectors. Recent empirical research underscores the urgency of this analysis: Hatami et al. [11] surveyed security threats targeting AI agents in cyber-physical systems and identified critical vulnerabilities at the MCP boundary, including context poisoning (where malicious tool outputs influence model behavior), supply-chain exposure from unvetted servers, and over-privileged credentials. Their findings — including tool poisoning detected in 5.5% of MCP servers tested — demonstrate that the protocol interfaces agents rely on are already being exploited. AXTP's experience pools introduce an additional attack surface that requires dedicated governance, as compromised experience compounds across agent networks rather than affecting a single session.

Threat	Severity	Primary Mitigation
Experience poisoning	High	Reputation system + anomaly detection
Trust amplification	High	Graph analysis + independence requirements
Knowledge drift	Medium	Temporal decay + re-validation
Data exfiltration	Medium	Access control + rate limiting + audit

Model extraction via XRs	Medium	Content summarization (no raw data)
Supply chain compromise	High	Agent identity verification + signing

Table 3: Primary threat vectors and mitigations.

5.2 Privacy Considerations

AXTP incorporates privacy-preserving design principles throughout the protocol. Experience Records contain only summarized inputs and outputs, never raw data from task execution. Organizations may configure field-level redaction policies to prevent sensitive information from entering pools. All records are encrypted at rest and in transit. Organizational pool isolation prevents cross-boundary information leakage. Retrieval rate limiting prevents systematic data exfiltration.

5.3 Compliance Alignment

AXTP's governance framework is designed with regulatory alignment in mind. The audit trail supports Article 12 (Record-keeping) and Article 14 (Human oversight) requirements of the EU AI Act. The trust scoring and validation mechanisms align with the NIST AI RMF's Govern and Measure functions. The access control and scope isolation features support data residency and sovereignty requirements. Organizations implementing AXTP should evaluate retention policies, right-to-erasure implications, and cross-boundary transfer rules within their specific regulatory context.

5.4 Empirical Evidence: MCP Boundary Vulnerabilities

Recent research provides empirical grounding for AXTP's governance-first design. Hatami et al. [11] present the SENTINEL framework, a lifecycle-aware methodology for evaluating security threats targeting AI agents in cyber-physical systems. Their survey identifies three categories of MCP boundary risk that directly inform AXTP's threat model:

Context Poisoning: Malicious tool outputs injected through MCP connections can influence downstream model behavior. In an AXTP context, this threat extends to experience pools: a poisoned Experience Record, if retrieved and applied by consuming agents, propagates the attacker's influence across the entire agent network. AXTP mitigates this through composite trust scoring, mandatory validation before high-trust status, and statistical anomaly detection on deposited XRs.

Supply-Chain Exposure: Hatami et al. report tool poisoning in 5.5% of MCP servers tested, with prompt injection exhibiting high prevalence. For experience pools, supply-chain risk manifests as compromised agents depositing subtly incorrect experience — more dangerous than overt poisoning because it may pass surface-level validation. AXTP addresses this through agent reputation tracking, outcome correlation scoring (do agents that consume this XR actually perform better?), and graph-based analysis to detect coordinated deposit patterns

from colluding agents.

Cascading Failure in Agent Networks: The survey documents how a single compromised agent can propagate failures through interconnected agent systems. Experience pools amplify this risk: one poisoned XR consumed by multiple agents creates correlated failures across the network. AXTP's temporal decay, re-validation triggers for high-consumption XRs, and consumer feedback loops are specifically designed to contain cascade propagation by rapidly degrading trust scores for XRs associated with negative downstream outcomes.

The SENTINEL framework's emphasis on lifecycle-aware security — integrating threat characterization, feasibility analysis under operational constraints, defense selection, and continuous validation — aligns with AXTP's approach to governance as a continuous process rather than a point-in-time check. Together, these findings validate AXTP's design principle that shared agent experience requires purpose-built governance infrastructure, not merely access controls layered onto existing storage systems.

6. Reference Implementation and Evaluation

6.1 Implementation Overview

A reference implementation of AXTP has been developed in Python, implementing the complete protocol including Experience Record creation and serialization, Experience Pool management with in-memory storage, all four core operations (Deposit, Retrieve, Validate, Inspect), the trust engine with composite scoring, and a demonstration scenario with multiple interacting agents.

The implementation is published as open source under the Apache 2.0 license at github.com/edwrdsr/axtp.

6.2 Demonstration Results

The reference implementation includes a demonstration scenario simulating two agents executing similar tasks with and without access to pooled experience:

Metric	Agent Alpha (No prior experience)	Agent Beta (With AXTP pool)
Task completion time	10,900 ms	3,850 ms
Steps executed	5 (incl. 1 failure)	2 (no failures)
Pivots required	1	0
Errors encountered	1	0
New knowledge contributed	2 patterns, 1 antipattern	1 new pattern

Table 4: Demonstration results comparing agent performance with and without AXTP pool access.

Agent Beta, with access to Agent Alpha's deposited experience, completed the task in 65% less time, executed fewer steps, required no pivots, and encountered no errors — while still contributing new knowledge (trial period configuration) to the pool. This demonstrates the core thesis: pooled experience enables compound intelligence where each agent's contribution improves outcomes for all subsequent agents.

Limitations: These results are from a controlled demonstration scenario, not a production benchmark. The agents are simulated, and the execution times represent estimated durations. Section 6.3 describes planned controlled experiments with live agent executions.

6.3 Benchmark Methodology (Planned)

A controlled benchmark is planned to validate AXTP's effectiveness with live agent executions. The methodology will include: selection of 3-5 common agent task categories (API integrations, data transformations, code debugging, document processing, multi-step reasoning); execution of each task category by 20+ agent instances, with a control group (no pool access) and treatment group (AXTP pool access); measurement of completion time, error rate, retry count, pivot frequency, and task success rate; and statistical analysis of performance differences. Results will be published in a subsequent version of this paper and in the project repository.

7. Discussion and Future Work

AXTP addresses a clear gap in the agent protocol landscape, but several open questions and opportunities for extension remain:

Semantic Retrieval: The current retrieval mechanism uses hierarchical task-type matching. Future implementations should incorporate embedding-based semantic similarity to match XRs based on contextual meaning rather than string matching. This would enable retrieval of relevant experience even when task taxonomies differ across organizations.

MCP Integration: An AXTP MCP server would expose Deposit and Retrieve as tools in the MCP ecosystem, enabling any MCP-compatible agent to participate in experience pools with minimal integration effort. This is the highest-priority implementation target.

Automated XR Generation: Agent framework integrations (LangChain callbacks, CrewAI hooks) could automatically generate XRs from agent execution traces, removing the need for explicit instrumentation and lowering the adoption barrier.

Cryptographic Verification: Experience Pool audit trails could be anchored to blockchain or distributed ledger systems for tamper-evident verification. XRs could be cryptographically signed by the producing agent, enabling provenance verification across trust boundaries. This aligns with emerging work on blockchain-based AI governance infrastructure.

Federated Pools: Organizations may wish to share experience across organizational boundaries without centralizing data. A federated pool architecture, where XRs remain in organizational storage but are discoverable and retrievable through a shared index, would address data sovereignty concerns while enabling cross-organizational learning.

Experience Summarization: As pools grow, retrieval efficiency becomes critical. Periodic summarization of accumulated XRs into condensed knowledge artifacts — preserving high-confidence patterns and antipatterns while archiving individual records — would maintain pool performance at scale.

8. Conclusion

The absence of a standardized protocol for agent experience transfer represents a significant infrastructure gap in the emerging agent ecosystem. While existing protocols address tool connectivity (MCP), task delegation (A2A), and message structure (ACP), none provide a mechanism for structured, validated, cross-agent experiential knowledge transfer.

AXTP addresses this gap by defining Experience Records as the atomic unit of transferable agent knowledge, Experience Pools as governed shared repositories, and a trust and governance framework that makes shared experience safe and reliable at scale. Initial demonstration results show significant efficiency gains when agents access pooled prior experience, with a 65% reduction in task completion time and elimination of redundant error exploration.

The protocol is published as an open specification to enable broad adoption, community scrutiny, and interoperability across the fragmented agent framework landscape. We invite the agent infrastructure community to review, implement, and extend AXTP, and to contribute to the critical work of defining how AI agents learn — not just individually, but collectively.

References

[1] Anthropic. (2024). Model Context Protocol Specification. https://modelcontextprotocol.io

[2] Google. (2025). Agent-to-Agent Protocol (A2A). https://google.github.io/A2A

[3] IBM. (2025). Agent Communication Protocol (ACP). https://agentcommunicationprotocol.dev

[4] AI Engineer Foundation. (2024). Agent Protocol: An Open API Specification. https://agentprotocol.ai

[5] CopilotKit. (2025). AG-UI: Agent-User Interaction Protocol. https://github.com/ag-ui-protocol/ag-ui

[6] Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S., Stoica, I., & Gonzalez, J. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.

[7] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. NeurIPS 2023.

[8] European Parliament. (2024). Regulation (EU) 2024/1689 (AI Act).

[9] National Institute of Standards and Technology. (2023). AI Risk Management Framework (AI RMF 1.0). NIST AI 100-1.

[10] Edwards, R. (2026). AXTP Reference Implementation. https://github.com/edwrdsr/axtp

[11] Hatami, M., Pham, V.T., Lakadawala, H., & Chen, Y. (2026). Securing AI Agents in Cyber-Physical Systems: A Survey of Environmental Interactions, Deepfake Threats, and Defenses. arXiv:2601.20184.